Coding for Privacy in Distributed Computing

Reent Schlegel

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2023

UNIVERSITY OF BERGEN

Coding for Privacy in Distributed Computing

Reent Schlegel

Thesis for the degree of Philosophiae Doctor (PhD)
at the University of Bergen

Date of defense: 21.04.2023

Year: 2023

Title: Coding for Privacy in Distributed Computing

Name: Reent Schlegel

Print: Skipnes Kommunikasjon / University of Bergen

Acknowledgements

My deepest gratitude goes to my “doctor fathers” Eirik and Àlex. The two of you are an excellent supervisor team and I count myself very lucky to have had you as professional and morale support. Special thanks also go to Sidd for the fruitful discussions and great collaboration.

Lena, your continued patience and support gave me the strength to finish this thesis.

And to my colleagues at Simula UiB, you made the office not only a place to work but rather a place to enjoy many fun moments at.

Thank you all!

Reent Schlegel
Bremen, December 2022

Abstract

In a distributed computing network, multiple devices combine their resources to solve a problem. Thereby the network can achieve more than the sum of its parts: cooperation of the devices can enable the devices to compute more efficiently than each device on its own could and even enable the devices to solve a problem neither of them could solve on its own. However, devices taking exceptionally long to finish their tasks can exacerbate the overall latency of the computation. This so-called straggler effect can arise from random effects such as memory access and tasks running in the background of the devices. The effect typically stalls the whole network because most devices must wait for the stragglers to finish. Furthermore, sharing data and results among devices can severely strain the communication network. Especially in a wireless network where devices have to share a common channel, e.g., in edge computing and federated learning, the communication links often become the bottleneck. Last but not least, offloading data to untrusted devices raises privacy concerns. A participant in the distributed computing network might be weary of sharing personal data with other devices without adequately protecting sensitive information.

This thesis analyses how ideas from coding theory can mitigate the straggler effect, reduce the communication load, and guarantee data privacy in distributed computing. In particular, Part A gives background on edge computing and federated learning, two popular instances of distributed computing, linear regression, a common problem to be solved by distributed computing, and the specific ideas from coding theory that are proposed to tackle the problems arising in distributed computing. Part B contains papers on the research performed in the framework of this thesis. The papers propose schemes that combine the introduced coding theory ideas to minimize the overall latency while preserving data privacy in edge computing and federated learning. The proposed schemes significantly outperform state-of-the-art schemes. For example, a scheme from Paper I achieves an 8% speed-up for edge computing compared to a recently proposed non-private scheme while guaranteeing data privacy, whereas the schemes from Paper II achieve a speed-up factor of up to 18 for federated learning compared to current schemes in the literature for considered scenarios.

Keywords: Coding Theory, Distributed Computing, Privacy, Straggler Mitigation

Abstrakt

I et distribuert datanettverk samarbeider flere enheter for å løse et problem. Slik kan vi oppnå mer enn summen av delene: samarbeid gjør at problemet kan løses mer effektivt, og samtidig blir det mulig å løse problemer som hver enkelt enhet ikke kan løse på egen hånd. På den annen side kan enheter som bruker veldig lang tid på å fullføre sin oppgave øke den totale beregningstiden betydelig. Denne såkalte straggler-effekten kan oppstå som følge av tilfeldige hendelser som minnetilgang og oppgaver som kjører i bakgrunnen på de ulike enhetene. Straggler-problemet blokkerer vanligvis hele beregningen siden alle enhetene må vente på at de treigeste enhetene blir ferdige. Videre kan deling av data og delberegninger mellom de ulike enhetene belaste kommunikasjonsnettverket betydelig. Spesielt i et trådløst nettverk hvor enhetene må dele en enkelt kommunikasjonskanal, for eksempel ved beregninger langs kanten av et nettverk (såkalte kantberegninger) og ved føderert læring, blir kommunikasjonen ofte flaskehalsen. Sist men ikke minst gir deling av data med upålitelige enheter økt bekymring for personvernet. En som ønsker å bruke et distribuert datanettverk kan være skeptisk til å dele personlige data med andre enheter uten å beskytte sensitiv informasjon tilstrekkelig.

Denne avhandlingen studerer hvordan ideer fra kodeteori kan dempe straggler-problemet, øke effektiviteten til kommunikasjonen og garantere datavern i distribuert databehandling. Spesielt gir del A en innføring i kantberegning og føderert læring, to populære instanser av distribuert databehandling, lineær regresjon, et vanlig problem som kan løses ved distribuert databehandling, og relevante ideer fra kodeteori. Del B består av forskningsartikler skrevet innenfor rammen av denne avhandlingen. Artiklene presenterer metoder som utnytter ideer fra kodeteori for å redusere beregningstiden samtidig som datavernet ivaretas ved kantberegninger og ved føderert læring. De foreslåtte metodene gir betydelige forbedringer sammenlignet med tidligere metoder i litteraturen. For eksempel oppnår en metode fra artikkel I en 8%-hastighetsforbedring for kantberegninger sammenlignet med en nylig foreslått metode. Samtidig ivaretar vår metode datavernet, mens den metoden som vi sammenligner med ikke gjør det. Artikkel II presenterer en metode som for noen brukstilfeller er opp til 18 ganger raskere for føderert læring sammenlignet med tidligere metoder i litteraturen.

Søkeord: Distribuert databehandling, kodeteori, personvern, straggler-demping

List of Papers

This thesis is based on the following publications:

Paper I
R. Schlegel, S. Kumar, E. Rosnes, A. Graell i Amat, *Privacy-Preserving Coded Mobile Edge Computing for Low-Latency Distributed Inference*, IEEE Journal on Selected Areas in Communications, vol. 40, no. 3, pp. 788-799, March 2022.

Paper II
R. Schlegel, S. Kumar, E. Rosnes, A. Graell i Amat, *CodedPaddedFL and CodedSecAgg: Straggler Mitigation and Secure Aggregation in Federated Learning*, submitted to IEEE Transactions on Communications (revised September and December 2022).

Other publications by the author which are not included in this thesis, but are included within the papers of the thesis, are:

- R. Schlegel, S. Kumar, E. Rosnes, A. Graell i Amat, *Private Edge Computing for Linear Inference Based on Secret Sharing*, in Proc. IEEE Global Commun. Conf. (GLOBECOM), Taipei, Taiwan, December 2020.
- S. Kumar, R. Schlegel, E. Rosnes, A. Graell i Amat, *Coding for Straggler Mitigation in Federated Learning*, in Proc. IEEE Int. Conf. Commun. (ICC), Seoul, South Korea, May 2022.
- R. Schlegel, S. Kumar, E. Rosnes, A. Graell i Amat, *Straggler-Resilient Secure Aggregation for Federated Learning*, in Proc. Eur. Sign. Process. Conf. (EUSIPCO), Belgrade, Serbia, August/September 2022.

Contents

List of Figures

Part A

Overview

Chapter 1

Background

1.1 Introduction

The increase in data throughout all aspects of everyday life ignited a move to decentralized data processing. A single traditional computing device would be overwhelmed by the amount of data that needs to be processed by modern applications. Therefore, distributed computing architectures were proposed, e.g., Google's patented MapReduce [1], that lead to powerful computing entities such as warehouse-scale computers (WSCs) [2]. These WSCs are a crucial enabler of modern cloud computing services.

While WSCs provide the computation power needed by many of nowadays's applications, they are not the ultimate solution to all data processing problems. Sending massive amounts of data for processing to far away computing centers can entail a prohibitive amount of latency or be unfeasible due to network limitations altogether. To mitigate the huge latency and reduce network traffic, the computational resources need to be moved closer to the devices generating the data that needs to be processed. Edge computing (EC) and federated learning (FL) are two distributed computing paradigms that process data close to the source. EC has become one of the pillars of the 5G mobile communication standard [3], whereas FL is used by industry-leading companies such as Google [4] and IBM [5].

In an EC network, the nodes at the network edge, e.g., the base stations in a cellular network, are equipped with computational resources. As a result, these so-called edge nodes (ENs) can provide additional services to customers, such as mobile phones in a cellular network and vehicles. For example, EC is a crucial enabler for autonomous driving which is depicted in Fig. 1.1. Data from multiple vehicles can be collected at the network edge for processing. Instead of performing computations locally on the vehicles with limited information, the computationally powerful ENs can gather information from all vehicles in the vicinity and perform computations to make recommendations for the vehicles in the network. Here, decisions on breaking and lane switching are highly delay critical and prohibit an offloading of the tasks to far away cloud computing centers. Similarly, cloud gaming is a latency-critical and data-intensive application that does not permit outsourcing to remote servers. On the other hand, remote sensing might not necessarily be latency critical; however, preprocessing sensor results can significantly reduce network traffic.

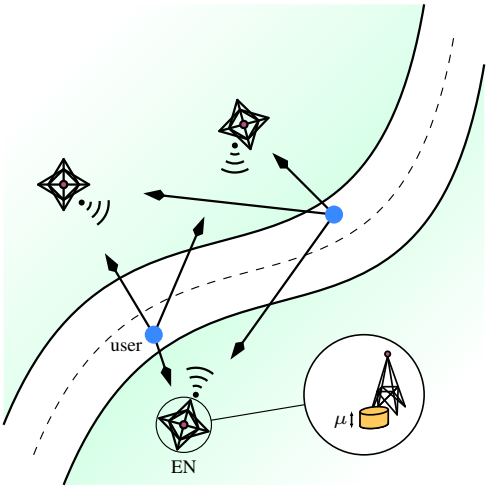


Figure 1.1: A mobile EC network applied to autonomous driving.

FL is a distributed learning framework in which multiple devices collaboratively train a global model on local data. Each device computes model updates on its local data and sends the computed updates to a central server. The central server aggregates the results from the devices and updates the global model. This process is iterated several times until the model converges. As a result, FL enables devices to jointly train a model without exchanging their local data. This keeping of local data makes FL especially attractive when several entities wish to infer a model on sensitive or private data. For example, new disease models can be trained in the healthcare sector without exchanging patients' data. Similarly, in finance, multiple businesses can jointly train a model without exchanging company confidential data.

While decentralization provides many benefits, as shown above, it also imposes new challenges. Load assignment and scheduling are non-trivial problems; however, in this work, I will focus on the *straggler problem*, the *communication bottleneck*, and *privacy*. Every computation incurs a random delay due to memory access and other tasks running in the background. This random delay can make waiting for the last device to finish its computation highly inefficient [6]. When the number of devices in a network becomes large, a few devices will take an exceptionally long time to finish their tasks, while the other devices will have to wait idly. This so-called straggler effect significantly impacts the overall latency of a distributed computing system.

Furthermore, offloading vast amounts of data, as could be the case in EC, can lead to network congestion. One of the main benefits of EC and FL is that there is no need to transmit massive amounts of data over long distances. Nevertheless, the (possibly wireless) links between the user devices and the ENs will become contested if no intelligent way of transmitting the data is utilized.

Lastly, offloading data for processing to possibly untrusted servers entails privacy concerns. Even in FL, where no private data is directly shared with anyone, the local model updates can contain information about the dataset used for train-

ing.

Erasure channel coding proved to be a successful tool in mitigating the *straggler problem* in distributed computing [7]. Introducing redundancy on the data prior to distributing it to the workers, i.e., the computing entities in a distributed computing scenario, enables the master, i.e., the entity offloading the computations to the distributed computing cluster, to recover the desired computation result from the results of a subset of the workers. Thereby, straggling workers can be ignored as it is sufficient to contact only the fastest workers. This ability to ignore slow workers is traded off with a slightly increased computational load for each worker. This idea has successfully been applied to matrix-vector and matrix-matrix multiplication [8–12], distributed learning [13], distributed optimization [14], as well as general polynomial computations [15]. In these works, maximum distance separable (MDS) codes are especially prominent candidates. They allow the master to recover the result by contacting any k out of n workers. However, the decoding complexity is often neglected. It was shown that the high decoding complexity of MDS codes, which usually scales quadratically with the code length, can nullify any benefit obtained through straggler mitigation if they are applied naively [31]. Rateless codes are a suitable alternative due to their low decoding complexity [16]. Furthermore, while they lack the MDS property, i.e., it is not necessarily sufficient to contact any k out of n workers, their ability to utilize subtasks from even the slow workers helps mitigate the straggler effect [17].

To avoid the *communication bottleneck* in the download of EC, [18, 19] proposed replicating tasks across different ENs. Thereby, the spatial diversity in the network is increased, and the ENs can utilize beamforming to serve multiple users simultaneously. [20–22] extended this idea by combining task replication for spatial diversity with MDS codes for straggler mitigation, again neglecting decoding complexity. [23] proposes a scheme combining a rateless code with irregular repetition which significantly reduces latency compared to the schemes in [20–22].

Privacy of users' data becomes an issue when naively offloading tasks to untrusted workers, such as in EC. To prevent workers from inferring private information, [24–27] propose schemes that secretly share the users' data among the workers such that any subset of z colluding workers cannot infer information about the users' data while mitigating the straggler effect. [24, 25] propose new schemes for linear computations, a flexible MDS like construction [24] and a combination of a rateless code with random padding [25], whereas [15, 27] consider non-linear computations by using Lagrange [15] and polynomial codes [27].

Moreover, even though no private data leaves the user devices in FL, model inversion attacks by the central server can leak information [28, 29]. The model updates at the devices reveal information about the local dataset used to train the model. When the central server collects sufficient updates throughout the epochs, it can use those to infer information about the local data at the user devices. To prevent model inversion attacks, secure aggregation protocols were proposed [30–42]. In secure aggregation, the central server only learns the aggregated model update of the devices and no single local model update. Thereby, the central server cannot infer information about any of the local datasets but rather only about the global dataset across all devices. The inferring of the global dataset by the central server is considered an acceptable level of privacy, as there usually are many

devices in an FL scenario and the contribution of any single user is well hidden among the masses.

In this thesis, I will present schemes highlighting the advantage coding provides in reducing the latency in both EC and FL. For EC, the scheme combines a product code for straggler mitigation with repetition for spatial diversity and Shamir's secret sharing [43] for user data privacy. Due to its lower decoding complexity, the scheme outperforms a previous non-private scheme from [20]. For FL, the first proposed scheme, CodedPaddedFL, combines gradient codes [13] for straggler mitigation with one-time padding for data privacy. It is the first scheme to apply coding to FL without leaking additional information to the central server. Furthermore, the second scheme for FL, CodedSecAgg, extends the ideas of CodedPaddedFL to secure aggregation to prevent the central server from launching a model inversion attack. CodedSecAgg outperforms state-of-the-art secure aggregation schemes such as [35] in terms of latency due to its straggler mitigation capabilities.

1.2 Outline

This chapter concludes with the notation's description. Chapter 2 introduces the particular problems both EC and FL face. In particular, the straggler effect is explained in detail, the communication bottleneck is addressed and why traditional multiple access strategies might be insufficient, and the particular privacy concerns are highlighted. In Chapter 3, linear regression is discussed. Linear regression is the central application of the schemes in this thesis. In the FL framework, we focus on the generation and learning of the model, whereas in the EC framework, we focus on the application of the learned model. Mitigation techniques for the problems outlined in Chapter 2 are explained in detail in Chapter 4. Chapter 5 gives a brief overview of the two papers considered in this thesis. The previously introduced ideas are put into context for each paper, and the main results are summarized. Lastly, Chapter 6 concludes Part A of the thesis. Part B includes the two papers *"Privacy-Preserving Coded Mobile Edge Computing for Low-Latency Distributed Inference"* and *"CodedPaddedFL and CodedSecAgg: Straggler Mitigation and Secure Aggregation in Federated Learning"* that are part of this thesis.

1.3 Notation

We use uppercase and lowercase bold letters for matrices and vectors, respectively, italics for sets, and sans-serif letters for random variables (RVs), e.g., $\boldsymbol{X}$, $\boldsymbol{x}$, $\mathcal{X}$, and X represent a matrix, a vector, a set, and an RV, respectively. An exception is $\hat{\mathsf{X}}$ in Section 3.4 which is a vector of RVs. Vectors are represented as row vectors throughout the chapter. For natural numbers m and n, $\mathbf{0}_{m\times n}$ denotes the all-zero matrix of size $m \times n$, $\mathbf{1}$ denotes the all-one matrix, and $\boldsymbol{I}$ denotes the identity matrix. The conjugate transpose of a matrix $\boldsymbol{X}$ is denoted as $\boldsymbol{X}^{\mathsf{H}}$ (or $\boldsymbol{X}^{\mathsf{T}}$ when $\boldsymbol{X}$ is over the reals). Furthermore, we represent the Euclidean norm of a vector $\boldsymbol{x}$ by $\|\boldsymbol{x}\|_2$, while the absolute value of a number x is $|x|$. $\Pr\{\mathsf{X} = x\}$ is the probability that a

realization of the RV X has the value x, while $\Pr\{\mathsf{X} = x|\mathsf{Y} = y\}$ is the conditional probability and $\mathbb{E}[\mathsf{X}]$ is the expected value of X. $\mathrm{I}(\cdot;\cdot)$ denotes mutual information and $\mathrm{H}(\cdot)$ and $\mathrm{H}(\cdot|\cdot)$ are the entropy and the conditional entropy, respectively. The gradient of a function $f(\boldsymbol{X})$ with respect to $\boldsymbol{X}$ is denoted by $\nabla_{\boldsymbol{X}} f(\boldsymbol{X})$. For a positive integer m, $[m]$ denotes the set $\{1, \ldots, m\}$ and for a real number r, $\lfloor r \rfloor$ is the largest integer less than or equal to r. The binomial coefficient is represented by $\mathrm{binom}\{n, k\}$. The finite field of size q is denoted by $\mathbb{F}_q$, whereas $\mathbb{R}$ and $\mathbb{C}$ denote the field of real numbers and complex numbers, respectively.

Chapter 2

Edge Computing and Federated Learning

EC and FL are distributed computing frameworks for wireless networks. As such, they have many similarities, including the challenges they face in their efficient execution. In this chapter, I will focus on three main challenges both EC and FL face. These are the straggler effect, the communication bottleneck, and guaranteeing users' data privacy.

2.1 Straggler Problem

The computation times incurred by the workers to perform their tasks are random. Memory access and tasks running in the background have a non-deterministic effect on the runtime. Usually, the computation times are modeled by a shifted randomly distributed RV. The shift represents the deterministic part of performing calculations in the circuitry, whereas the exponential tail in the distribution pertains to random events such as memory access and background tasks. When a large computation is naively distributed across multiple workers, the master must wait for the result of all workers —including the slowest worker —to obtain the desired result. As a result, the waiting time at the master when distributing the task across n workers is the n-th order statistic of a sample of n shifted exponentially distributed RVs. Let us assume the workers' computation times λ_i are independent and identically distributed and let us view the shifted exponential variable as the sum of a deterministic real value s and an exponentially distributed RV λ'_i with parameter η, i.e., $\lambda_i = s + \lambda'_i$ with $\mathbb{E}\{\lambda'_i\} = 1/\eta$. Then, the n-th order statistic $\lambda_{(n)}$ of $\{\lambda_i\}$ is equal to $\lambda_{(n)} = s + \lambda'_{(n)}$, where $\lambda'_{(n)}$ is the n-th order statistic of $\{\lambda'_i\}$. Alfréd Rényi derived the general expression for the k-th order statistic $\lambda_{(k)}$ of an n-sized sample of exponentially distributed RVs in [44]:

$$\lambda_{(k)} = \frac{1}{\eta}\left(\sum_{j=1}^{k} \frac{Z_j}{n-j+1}\right), \tag{2.1}$$

where the Z_js are standard exponential RVs. As a result, we have

$$\mathbb{E}\{\lambda_{(k)}\} = \frac{1}{\eta}\left(\sum_{j=1}^{k} \frac{1}{n-j+1}\right). \tag{2.2}$$

To highlight this so-called *straggler effect*, I will use the following example. Let $s = 0$ and $\eta = 1$, i.e., let the computation times at the workers be purely exponentially distributed with expected value 1, and let there be $n = 10$ workers. Although the expected computation time of each worker is only 1 time unit, the expected computation time of the slowest out of the 10 workers is approximately 2.93, almost 3 times slower than the average computation time. We can see that even for a small number of workers, the straggler effect severely impacts the master's waiting time.

2.2 Communication Bottleneck

Both in EC and FL, we have multiple devices communicating with each other simultaneously through a shared wireless channel. Without proper access control, the devices will interfere with each other, rendering all communication impossible. Traditionally, different devices are assigned different time slots, so-called time division multiple access (TDMA), or different frequency bands, so-called frequency division multiple access (FDMA) to guarantee that different devices do not

interfere with each other. In TDMA, the channel is used by only one device at any given time, whereas in FDMA, all devices can transmit simultaneously, and the receiver can recover the desired transmission through the use of an appropriate bandpass filter. A combination of TDMA and FDMA is also possible. However, as the number of devices increases, these access techniques are no longer suited. The need for guard intervals, both in time and frequency, leaves no usable time slots or frequency bands as the number of needed slots/bands increases.

Space (SDMA) and code division multiple access (CDMA) are alternative access techniques. In SDMA, the network is spatially partitioned into smaller networks. This partitioning allows the reuse of time slots and frequency bands in spatially separated parts of the network. However, SDMA requires directional antennas to prevent interference across different spatial partitions. In practice, the directionality of reasonably priced antennas is limited, which leads to only a small number of different partitions. In cellular networks, for example, each base station partitions the assigned space into three sectors with a single 120° antenna each. CDMA assigns a code $\boldsymbol{c}_i$ to device i for each device in the network. These codes are orthonormal, meaning that we have $\langle \boldsymbol{c}_i, \boldsymbol{c}_i \rangle = 1$ and $\langle \boldsymbol{c}_i, \boldsymbol{c}_j \rangle = 0$ for $i \neq j$. Device i transmits the product of the message m_i it wants to send and the assigned code $\boldsymbol{c}_i$, i.e., device i transmits $\boldsymbol{x}_i = m_i \boldsymbol{c}_i$. In a noiseless channel, the receiver observes $\boldsymbol{y} = \sum_i \boldsymbol{x}_i$, the superposition of all transmitted messages. The receiver recovers the desired message m_j as $m_j = \langle \boldsymbol{y}, \boldsymbol{c}_j \rangle$. The cost of CDMA is a bandwidth expansion due to the need to accommodate many orthonormal codes.

All four traditional access technologies have worked well in the past but are not suited for a very large number of devices in a network. The need for guard bands in TDMA and FDMA renders those techniques inefficient for a huge number of devices, whereas SDMA has hardware limitations that hinder unlimited partitioning of the network, and CDMA suffers from a considerable bandwidth expansion for a large number of devices. This means new techniques are needed to facilitate a huge number of devices that simultaneously want to communicate, as is the case in EC and FL.

2.3 Privacy

Offloading data for processing to untrusted devices entails huge risks to data privacy. In EC, users need to offload their data to servers at the network's edge. While users may or may not trust these ENs' provider(s), untrusted malicious third parties might compromise the servers. Due to the distributed nature in all kinds of environments, it is virtually impossible to guarantee the cyber and physical security of the ENs. As a result, an adversary might gain access to several servers belonging to the (maybe trusted) service providers. Consequently, even when users are willing to reveal their personal data to a service provider, there is no guarantee that their data remains only visible to the provider. Hence, there is a need for a privacy-preserving protocol to process users' data that is resilient against an adversary with access to several servers.

In FL, the devices process their own data locally, thereby seemingly retaining the privacy of their data. However, the local model updates contain information

about the local datasets used to train them. Hence, by observing the model updates, the central server can infer information about the local data. In the worst case, the model updates can reveal the whole dataset. Therefore, it is vital to prevent these *model inversion attacks*. Furthermore, as we will see later, it can be beneficial to introduce data redundancy across different devices to mitigate the straggler effect. Naively introducing this redundancy will leak information about the devices' private data to other untrusted devices.

Chapter 3

Linear Regression

Linear models play an essential role in many facets of life. Trend lines, for example, are commonly used to predict or analyze markets and populations, whereas the capital asset pricing model is a widely used linear model in finance and the linear regression algorithm is one of the fundamental building blocks in machine learning. In this chapter, I will introduce the background for linear regression and how to obtain a model from sampled data.

3.1 Background

Linear regression models a linear dependence of a set of RVs $\mathcal{Y}$ on another set of RVs $\mathcal{X}$. Let $|\mathcal{X}| = n$ and $|\mathcal{Y}| = m$ and let $x_i \in \mathbb{M}$ and $y_j \in \mathbb{M}$ for $i \in [n]$ and $j \in [m]$ be realizations of the RVs $\mathsf{X}_i \in \mathcal{X}$ and $\mathsf{Y}_j \in \mathcal{Y}$ over some measurable space $\mathbb{M}$. In this work, we restrict $\mathbb{M}$ to be a ring as we are interested in the linear dependence of $\mathcal{Y}$ on $\mathcal{X}$, i.e., we need $\mathbb{M}$ to be closed under additions and multiplications. We collect $\{x_i | i \in [n]\}$ and $\{y_j | j \in [m]\}$ in the n-dimensional row vector $\boldsymbol{x} \in \mathbb{M}^n$ and m-dimensional row vector $\boldsymbol{y} \in \mathbb{M}^m$. We can then represent the linear dependence through a matrix $\boldsymbol{\Theta} \in \mathbb{M}^{n \times m}$ as

$$\boldsymbol{y} = \boldsymbol{x}\boldsymbol{\Theta}.$$

The matrix $\boldsymbol{\Theta}$ is often referred to as the model. Given the model $\boldsymbol{\Theta}$, one can now predict the outcome of the RVs in $\mathcal{Y}$ by observing the RVs in $\mathcal{X}$. This makes linear regression such a powerful tool. What remains to be shown is how to obtain the model $\boldsymbol{\Theta}$ from possibly noisy observations of $\mathcal{X}$ and $\mathcal{Y}$.

3.2 Obtaining the Model

To determine the model $\boldsymbol{\Theta}$, one has to sample the RVs in $\mathcal{X}$ and simultaneously observe the RVs in $\mathcal{Y}$. In practice, the observations might be corrupted by noise. Let $\{\hat{\boldsymbol{x}}_i | i \in [s]\}$ be the sample of size s of the RVs in $\mathcal{X}$ and $\{\hat{\boldsymbol{y}}_i | i \in [s]\}$ be the corresponding observations of the RVs in $\mathcal{Y}$. We can then express $\hat{\boldsymbol{y}}_i$ as

$$\hat{\boldsymbol{y}}_i = \hat{\boldsymbol{x}}_i\boldsymbol{\Theta} + \boldsymbol{n}_i,$$

where $\boldsymbol{n}_i$ is the measurement noise while observing $\hat{\boldsymbol{y}}_i$. We can collect the samples, observations, and noise as matrices

$$\hat{\boldsymbol{X}} = \begin{bmatrix} \hat{\boldsymbol{x}}_1 \\ \vdots \\ \hat{\boldsymbol{x}}_s \end{bmatrix}, \hat{\boldsymbol{Y}} = \begin{bmatrix} \hat{\boldsymbol{y}}_1 \\ \vdots \\ \hat{\boldsymbol{y}}_s \end{bmatrix}, \text{and } \boldsymbol{N} = \begin{bmatrix} \boldsymbol{n}_1 \\ \vdots \\ \boldsymbol{n}_s \end{bmatrix}$$

such that we can express their dependence as

$$\hat{\boldsymbol{Y}} = \hat{\boldsymbol{X}}\boldsymbol{\Theta} + \boldsymbol{N}.$$

A common estimate $\hat{\boldsymbol{\Theta}}_{\mathsf{LS}}$ of the true model $\boldsymbol{\Theta}$ is obtained by minimizing the squared error. For example, when $\boldsymbol{n}_i$ contains independent and identically distributed Gaussian RVs, the least squared error (LSE) estimator is equivalent to the maximum likelihood (ML) estimator. Note that $\boldsymbol{n}_i = \hat{\boldsymbol{y}}_i - \hat{\boldsymbol{x}}_i\boldsymbol{\Theta}$. Therefore, the estimate minimizing the LSE is given as

$$\hat{\boldsymbol{\Theta}}_{\mathsf{LS}} = \arg\min_{\tilde{\boldsymbol{\Theta}}} \frac{1}{2s} \sum_{i=1}^{s} \|\boldsymbol{n}_i\|_2^2 = \arg\min_{\tilde{\boldsymbol{\Theta}}} \frac{1}{2s} \sum_{i=1}^{s} \|\hat{\boldsymbol{y}}_i - \hat{\boldsymbol{x}}_i\tilde{\boldsymbol{\Theta}}\|_2^2. \tag{3.1}$$

The estimate $\hat{\boldsymbol{\Theta}}_{\mathsf{LS}}$ minimizes the squared error of a particular observation $\left(\hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}}\right)$. This can be a desired characteristic when one typically observes similar realizations of the RVs in $\mathcal{X}$ to the ones observed in $\hat{\boldsymbol{X}}$ and a tight fit of the model to the

sample $\hat{\boldsymbol{X}}$ is helpful. However, this high dependence on the specific realization $\hat{\boldsymbol{X}}$ and overfitting of the model can be detrimental when one expects to observe new data in the future and tries to extrapolate to make predictions with samples that are not contained in the $\hat{\boldsymbol{X}}$ that was used to derive the model $\hat{\boldsymbol{\theta}}_{\mathsf{LS}}$. From a Bayesian point of view, this is equivalent to the fact that the ML estimator is not optimal. Instead, one should include a prior model distribution to obtain a maximum a posteriori (MAP) estimator. Different objective functions for the model estimation arise depending on the a priori distribution that is used. The normal and Laplace distributions are the most widely used a priori distributions, leading to the so-called ridge regression and lasso regression estimates. The corresponding model estimations $\hat{\boldsymbol{\theta}}_{\mathsf{RR}}$ and $\hat{\boldsymbol{\theta}}_{\mathsf{LASSO}}$ are given as

$$\hat{\boldsymbol{\theta}}_{\mathsf{RR}} = \arg\min_{\tilde{\boldsymbol{\theta}}} \frac{1}{2s} \sum_{i=1}^{s} \|\hat{\boldsymbol{y}}_i - \hat{\boldsymbol{x}}_i \tilde{\boldsymbol{\Theta}}\|_2^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{m} \tilde{\theta}_{ij}^2 \tag{3.2}$$

and

$$\hat{\boldsymbol{\theta}}_{\mathsf{LASSO}} = \arg\min_{\tilde{\boldsymbol{\theta}}} \frac{1}{2s} \sum_{i=1}^{s} \|\hat{\boldsymbol{y}}_i - \hat{\boldsymbol{x}}_i \tilde{\boldsymbol{\Theta}}\|_2^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{m} |\tilde{\theta}_{ij}|. \tag{3.3}$$

We can view (3.2) to be the Lagrangian form of the optimization problem of the LSE estimator in (3.1) with the additional constrained that the 2-norm of the model $\hat{\boldsymbol{\theta}}_{\mathsf{RR}}$ is less than a threshold t, where the relationship between t and λ depends on the data. Similarly, (3.3) is equivalent to (3.1) with the constraint that the 1-norm of $\hat{\boldsymbol{\theta}}_{\mathsf{LASSO}}$ is less than a threshold t.

The benefit of the ridge regression estimate is that it reasonably assumes that the a priori distribution of the model is a normal distribution rather than the more exotic Laplace distribution. This makes ridge regression a suited candidate for a wide range of applications. However, the hypercubic shape of the constraint in LASSO leads to more coefficients in $\hat{\boldsymbol{\theta}}_{\mathsf{Lasso}}$ being equal to zero. That means a model estimate obtained through (3.3) selects a subset of the RVs in $\mathcal{X}$ that have a non-negligible influence on the prediction of $\mathcal{Y}$. This can lead to significantly simplified models as many of the RVs in $\mathcal{X}$ might be ignored for predicting $\mathcal{Y}$. The model $\hat{\boldsymbol{\theta}}_{\mathsf{Lasso}}$ will automatically discard the negligible values of $\mathcal{X}$ whereas $\hat{\boldsymbol{\theta}}_{\mathsf{RR}}$ most likely will consider a small but non-zero influence of these RVs.

It remains to be shown how to solve the optimization problems in (3.1) to (3.3).

3.3 Gradient Descent

There can exist closed-form solutions to the optimization problems for the model estimations. For example, the closed form solution to (3.1) is $\hat{\boldsymbol{\theta}}_{\mathsf{LS}} = \left(\hat{\boldsymbol{X}}^{\mathsf{T}} \hat{\boldsymbol{X}}\right)^{-1} \hat{\boldsymbol{X}}^{\mathsf{T}} \hat{\boldsymbol{Y}}$. However, the inverse might not exist, and when it does, it is expensive to compute and numerically unstable. Therefore, in most practical applications, the optimization problems are solved via gradient descent. The gradient $\nabla_{\boldsymbol{x}} f(\boldsymbol{p})$ at a point $\boldsymbol{p}$ of a multivariate differentiable function $f(\boldsymbol{x})$ points in the direction of steepest ascend of f around $\boldsymbol{p}$. Given an initial point $\boldsymbol{x}_0$, taking a small step in the opposite direction of the gradient, i.e., in the direction of the steepest descent of f at

$\boldsymbol{x}_0$, yields a new point $\boldsymbol{x}_1$ that will have a lower value $f(\boldsymbol{x}_1)$ than the initial value $f(\boldsymbol{x}_0)$. More precisely, for

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - \mu \cdot \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0)$$

with a sufficiently small step size μ, we have

$$f(\boldsymbol{x}_1) \leq f(\boldsymbol{x}_0).$$

Given an initial guess $\boldsymbol{x}_0$ of a local minimum, one can iteratively update the guess to find points that yield smaller and smaller values of the function f. When f fulfills certain criteria, for example, when f is convex, and the step size μ is chosen appropriately at each iteration, for example, through the line search algorithm, the gradient descent algorithm is guaranteed to converge to a local minimum. Furthermore, in the case where f is convex, every local minimum is the global minimum, i.e., for a convex and differentiable function $f(\boldsymbol{x})$, the gradient descent algorithm with appropriately chosen step size μ is guaranteed to converge to the global minimum.

Each of the objective functions in (3.1) to (3.3) is convex and differentiable, except (3.3) at $\boldsymbol{\theta} = \mathbf{0}$. This means we can use gradient descent to solve each optimization problem as long as we do not go through $\boldsymbol{\theta} = \mathbf{0}$ for the LASSO regression. For example, in the case of ridge regression, we have

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \frac{1}{m} \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{X}^\top \boldsymbol{Y} + 2\lambda\boldsymbol{\theta}.$$

Given an initial guess $\boldsymbol{\theta}_0$ of the model, we can iteratively update $\boldsymbol{\theta}_i$ at iteration i as

$$\boldsymbol{\theta}_i = \boldsymbol{\theta}_{i-1} - \mu_i \cdot \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_{i-1}),$$

where μ_i is the step size at iteration i. For a properly chosen μ_i we are guaranteed that $\lim_{i\to\infty} \boldsymbol{\theta}_i = \hat{\boldsymbol{\theta}}_{\mathrm{RR}}$.

This gives a computationally efficient way of finding a good model for the linear dependence of $\mathcal{Y}$ on $\mathcal{X}$. Next, we will discuss ways to apply linear regression to non-linear problems.

3.4 Application to Non-Linear Problems

Linear regression is very attractive due to its low computational complexity for obtaining the model. However, many problems, i.e., relations between $\mathcal{Y}$ and $\mathcal{X}$, are not linear. In some cases, a transformation of the input $\mathcal{X}$ can convert the problem into a linear relation between $\mathcal{Y}$ and the transformed $\hat{\mathcal{X}}$. For example, suppose we have a relation $y = f(x) = f_d x^d + f_{d-1} x^{d-1} + \cdots + f_1 x + f_0$ where $f(\cdot)$ is a polynomial of degree d. For $d > 1$ this represents a non-linear relation. Nevertheless, taking $\hat{\mathrm{X}} = (\mathrm{X}^d, \mathrm{X}^{d-1}, \cdots, \mathrm{X}, 1)^\top$ and $\boldsymbol{f} = (f_d, f_{d-1}, \cdots, f_1, f_0)^\top$ lets us express y as $y = \boldsymbol{f}^\top \hat{\boldsymbol{x}}$. Consequently, by transforming X we turned the non-linear dependency of Y on X into a linear dependency of Y on $\hat{\mathrm{X}}$. This transformation lets us apply the above-described techniques to infer $\boldsymbol{f}$ and consequently $f(\cdot)$.

This particular example applies only to polynomial models. However, some transformations linearize general non-linear problems. One such transformation is kernel embedding. Kernel methods rely on the fact that any positive definite function $k : \mathbb{M}^n \times \mathbb{M}^n \mapsto \mathbb{R}$ implicitly defines a mapping $\phi : \mathbb{M}^n \mapsto \mathcal{H}$ to a Hilbert space $\mathcal{H}$ such that $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$, the so-called kernel trick. That means we can apply models that only depend on the inner products of input arguments, i.e., linear models, to possibly highly non-linear functions $k(\boldsymbol{x}, \boldsymbol{x}')$ by previously applying ϕ to the inputs. Note that ϕ can have large, possibly infinite, dimensionality compared to $\mathbb{M}^n$. This high dimensionality can render kernel methods impractical during the model's training in practice. Furthermore, a possibly more efficient pre-computation/sampling of $k(\boldsymbol{x}, \boldsymbol{x}')$ for all pairs $(\boldsymbol{x}, \boldsymbol{x}')$ quickly becomes impractical for large datasets as the complexity scales quadratically in the number of distinct data points.

The authors in [45] propose to use an explicit randomized mapping $\boldsymbol{z}(\boldsymbol{x})$ to a low-dimensional Euclidean space instead of the implicit mapping ϕ. The authors show that inner products of $\boldsymbol{z}$ approximate the kernel evaluation k. In particular, the authors show that

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle \approx \boldsymbol{z}(\boldsymbol{x})^\top \boldsymbol{z}(\boldsymbol{x}').$$

Due to the low dimensionality of $\boldsymbol{z}$, fast linear models can be used to approximate the non-linear behavior of k. The method in [45] enjoys such high popularity that it was implemented in Python's sklearn package, and we utilize it in Paper II to apply linear regression to a non-linear classification problem.

Chapter 4

Mitigation Techniques

In this chapter, I will introduce techniques to mitigate or solve the problems outlined in Chapter 2, i.e., the straggler effect, communication bottleneck, and privacy concerns.

4.1 Reed-Solomon Codes

Reed-Solomon (RS) codes are a class of cyclic block codes introduced in 1960 [46]. RS codes encode a length k message with symbols from the finite field $\mathbb{F}_q$ of size q into a length n codeword. In particular, an (n, k, q) RS code with $k \leq n \leq q$ encodes a message $\boldsymbol{m} = (m_1, \ldots, m_k) \in \mathbb{F}_q^k$ into a codeword $\boldsymbol{c} = (c_1, \ldots, c_n) \in \mathbb{F}_q^n$. There are many ways to map from the set of all messages to the set of codewords, i.e., the encoding procedure is not unique. Here, we will focus on the map initially proposed by Reed and Solomon. This map constructs a degree $k-1$ polynomial over $\mathbb{F}_q$ with coefficients given by $\boldsymbol{m}$, which is evaluated at n distinct points to obtain the n codeword symbols. More precisely, we define

$$p_{\boldsymbol{m}}(x) = \sum_{i=1}^{k} m_i x^{i-1} = m_1 + m_2 x + m_3 x^2 + \cdots + m_k x^{k-1} \tag{4.1}$$

and evaluate $p_{\boldsymbol{m}}(x)$ at n distinct points. For example, let α be a primitive element of $\mathbb{F}_q$. We can then evaluate $p_{\boldsymbol{m}}(x)$ at points $\{0, 1, \alpha, \alpha^2, \ldots\}$, or, when $n < q$, at points $\{1, \alpha, \alpha^2, \ldots\}$. In this work, I will focus on the case with $n < q$, for which the reason will become clear later. We can then obtain the codeword as

$$\boldsymbol{c} = (p_{\boldsymbol{m}}(\alpha^0), p_{\boldsymbol{m}}(\alpha^1), \ldots, p_{\boldsymbol{m}}(\alpha^{n-1})). \tag{4.2}$$

Being a linear transformation of $\boldsymbol{m}$, the map in (4.2) can be represented with an encoding matrix $\boldsymbol{G}$. In particular, for

$$\boldsymbol{G} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \alpha^1 & \cdots & \alpha^{n-1} \\ 1 & (\alpha^1)^2 & \cdots & (\alpha^{n-1})^2 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & (\alpha^1)^{k-1} & \cdots & (\alpha^{n-1})^{k-1} \end{pmatrix} \tag{4.3}$$

we have $\boldsymbol{c} = \boldsymbol{m}\boldsymbol{G}$. Notice that $\boldsymbol{G}$ is (the transpose of) a Vandermonde matrix, meaning that any $k \times k$ submatrix of $\boldsymbol{G}$ has full rank. Therefore, any $k \times k$ submatrix of $\boldsymbol{G}$ is invertible, leading to the MDS property of RS codes.

4.1.1 Decoding

Any k correctly received codeword symbols c_i are sufficient to recover $\boldsymbol{m}$. Recall that $p_{\boldsymbol{m}}(x)$ is a degree $k-1$ polynomial, uniquely defined by any k points. This means that for any $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \geq k$ the elements of the set $\{c_i | i \in \mathcal{I}\} = \{p_{\boldsymbol{m}}(\alpha^{i-1}) | i \in \mathcal{I}\}$ can be used in a polynomial interpolation to recover $p_{\boldsymbol{m}}(x)$ which gives $\boldsymbol{m}$. For example, using the Lagrange polynomial for interpolation and restricting $|\mathcal{I}| = k$ one gets

$$p_{\boldsymbol{m}}(x) = \sum_{i \in \mathcal{I}} c_i * \prod_{j \in \mathcal{I}, j \neq i} \frac{x - \alpha^{j-1}}{\alpha^{i-1} - \alpha^{j-1}}.$$

Determining whether a coded symbol is error-free is straightforward at the output of an erasure channel. The symbols are either received correctly or lost. Consequently, every available symbol at the decoder is a correctly received symbol. Generally, when the channel introduces errors, it is more complicated. In this case, the decoder first has to determine which symbols are affected by errors (usually by computing an error locator polynomial); however, this is out of scope for this work as we will focus on the erasure channel.

4.1.2 Application to Distributed Computations

Suppose a master wants to compute the matrix-vector multiplication $\boldsymbol{A}\boldsymbol{x}$ with the help of three workers. The master can interpret $\boldsymbol{A}$ as $\boldsymbol{A} = [\boldsymbol{A}_1^\top, \boldsymbol{A}_2^\top, \boldsymbol{A}_3^\top]^\top$ and distribute one $\boldsymbol{A}_i$ to each of the three workers. Each worker then computes $\boldsymbol{A}_i\boldsymbol{x}$ and the master can recover the result $\boldsymbol{A}\boldsymbol{x} = [(\boldsymbol{A}_1\boldsymbol{x})^\top, (\boldsymbol{A}_2\boldsymbol{x})^\top, (\boldsymbol{A}_3\boldsymbol{x})^\top]^\top$ from the local results of all three workers. In this case, each worker performs one-third of the total computation load.

Alternatively, the master can interpret $\boldsymbol{A}$ as $\boldsymbol{A} = [\tilde{\boldsymbol{A}}_1^\top, \tilde{\boldsymbol{A}}_2^\top]^\top$ and apply a $(3, 2)$ RS code on the submatrices prior to distributing them. In particular, one worker gets $\tilde{\boldsymbol{A}}_1 + \tilde{\boldsymbol{A}}_2$, another $\tilde{\boldsymbol{A}}_1 + 2\tilde{\boldsymbol{A}}_2$, and the last worker $\tilde{\boldsymbol{A}}_1 + 3\tilde{\boldsymbol{A}}_2$. Again, each worker multiplies its assigned matrix by $\boldsymbol{x}$. This time, however, the master will be able to recover the desired result from the local results of any 2 of the workers. The downside of this approach is that each worker has to perform computations equivalent to half the overall computation load.

This idea is easily extended to the case for general n and k. For n workers, splitting $\boldsymbol{A}$ into k submatrices and applying an (n, k) RS code results in each worker performing tasks of size $1/k$ of the total computation load while the master has to wait for only the fastest k out of the n workers.

4.2 Gradient Codes

Gradient codes [13] are a particular class of codes in that they are tailored for distributed gradient descent. In distributed gradient descent, a master wishes to compute gradients on a dataset with the help of multiple devices. The master distributes the dataset across the devices, which in turn compute local gradients on the assigned part of the data. The master can then aggregate the local gradients to obtain the global gradient and perform a model update for the next iteration. Like any other distributed computation, distributed gradient descent suffers from the straggler effect. To mitigate the straggling problem, the master can apply gradient codes to introduce redundancy such that a subset of the devices' results is sufficient for the master to achieve its goal.

In contrast to traditional codes such as RS codes described above, using gradient codes does not necessarily allow the recovery of the initial uncoded gradients. In the case of distributed gradient descent, for example, using an (n, k) RS code, the master could recover k local gradients from the result of any k out of n devices. On the other hand, the master has no guarantee to recover any local gradient using gradient codes. However, the master is not interested in the local gradients.

The master's goal is to obtain the global gradient to update the model for the next iteration. Gradient codes enable the master to do precisely that.

Gradient codes are defined by two matrices, an encoding matrix $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ and a decoding matrix $\boldsymbol{A} \in \mathbb{R}^{s \times n}$, where n is the number of workers taking part in the gradient computation, and s is the number of straggling patterns that the code is designed to handle. For example, suppose the master wishes to recover the global gradient by contacting any k out of the n workers, then $s = \text{binom}\{n, k\}$. Furthermore, each row of $\boldsymbol{A}$ contains at most k non-zero elements, whereas each row of $\boldsymbol{B}$ contains at least $n - k + 1$ non-zero elements. There are multiple ways to construct $\boldsymbol{A}$ and $\boldsymbol{B}$; however, all have the following requirement in common:

$$\boldsymbol{A}\boldsymbol{B} = \mathbf{1}. \tag{4.4}$$

This requirement means that the span of any k rows of $\boldsymbol{B}$ contains the all-one vector and $\boldsymbol{A}$ contains the coefficients of the linear combination of any k rows of $\boldsymbol{B}$ to obtain the all-one vector.

4.2.1 Application to Distributed Gradient Descent

In a network with n workers a master wishes to perform distributed gradient descent on a global dataset $\mathcal{X}$. The master can divide $\mathcal{X}$ into n parts $\mathcal{X}_1, \cdots, \mathcal{X}_n$ each with their own partial gradient $\boldsymbol{g}_i$. In uncoded distributed gradient descent, each worker computes one partial gradient $\boldsymbol{g}_i$ on its assigned partition $\mathcal{X}_i$ and the master has to wait for all workers to finish their computation before obtaining the sum of all gradients $\sum_i \boldsymbol{g}_i$.

Using gradient codes, it is sufficient for the master to wait for the result of the k fastest workers out of the n available workers. This benefit comes at the cost of each worker having to compute multiple gradients. In particular, $\boldsymbol{A}$ and $\boldsymbol{B}$ are constructed with parameters n and k as shown in [13]. Then, worker i computes the gradients $\boldsymbol{g}_j$ given by the indices j of the nonzero elements in row i of $\boldsymbol{B}$. After finishing the assigned gradient computations, each worker i encodes the gradients by taking a linear combination with coefficients given by row i of $\boldsymbol{B}$. After receiving the first k gradients, the master uses the row of $\boldsymbol{A}$ with the non-zero coefficients corresponding to the k fastest workers and takes another linear combination of the received coded gradients given by the row of $\boldsymbol{A}$. From (4.4) it follows that the master indeed obtained the sum of all gradients $\sum_i \boldsymbol{g}_i$.

In case the partial gradient computation is linear in the corresponding partial dataset, the workers can also encode the $n - k + 1$ assigned datasets using their row of $\boldsymbol{B}$ and compute the encoded gradient on the encoded dataset. Due to the linearity of the gradient codes and the gradient computation the resulting coded gradients will be equivalent. In this case, the workers do not have to compute several gradients in each iteration but rather just one. This is traded off by the workers having to perform the encoding on the datasets once prior to the iterative gradient computations.

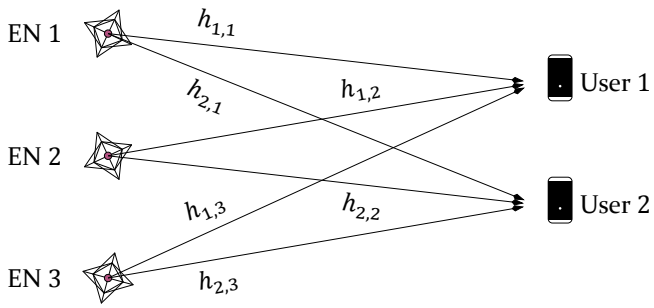


Figure 4.1: An example MIMO communication system with $N_t = 3$ transmit antennas and $N_r = 2$ receive antennas.

4.3 Multiuser Multiple Input Multiple Output Transmission (MIMO)

4.3.1 Multiplexing Gain

We consider a narrow-band communication system with N_t transmit antennas and N_r receiver antennas. Assuming each of the antennas is sufficiently far spaced apart (about half of the wavelength for omnidirectional antennas in a uniform scattering environment), the paths from each of the transmitter to each of the receiver antennas experience independent fading, and the whole transmission through the channel can be modeled as

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n},$$

where $\boldsymbol{H} \in \mathbb{C}^{N_r \times N_t}$ is the channel gain matrix, $\boldsymbol{x}$ is the transmitted signal, with x_i being transmitted on antenna i, $\boldsymbol{n}$ the additive white Gaussian noise (AWGN) at the receiver, and $\boldsymbol{y} \in \mathbb{C}^{N_r}$ the received signal with y_i being the signal received at antenna i. The entry $h_{i,j}$ in the channel gain matrix $\boldsymbol{H}$ pertains to the channel gain from transmit antenna j to receiver antenna i. An example is depicted in Fig. 4.1.

We assume that $\boldsymbol{H}$ is known both at the receiver and transmitter. When the channel stays constant sufficiently long, the receiver can easily obtain $\boldsymbol{H}$ by observing known pilots sent by the transmitter. Subsequently, the receiver can forward $\boldsymbol{H}$ through a feedback channel to the transmitter. Once $\boldsymbol{H}$ is known at transmitter and receiver, the channel can be decomposed into r orthogonal virtual channels with $r = \operatorname{rank}(\boldsymbol{H})$ as follows.

Let

$$\boldsymbol{H} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{H}}$$

be a singular value decomposition of $\boldsymbol{H}$, where $\boldsymbol{U} \in \mathbb{C}^{N_r \times N_r}$ and $\boldsymbol{V} \in \mathbb{C}^{N_t \times N_t}$ are unitary matrices[1] and

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{S} & \boldsymbol{0}_{r\times(N_t-r)} \\ \boldsymbol{0}_{(N_r-r)\times r} & \boldsymbol{0}_{(N_r-r)\times(N_t-r)} \end{pmatrix},$$

[1] A unitary transformation preserves the inner product of vectors. It follows for the matrix $\boldsymbol{U}$ of a unitary transformation that $\boldsymbol{U}^{\mathsf{H}}\boldsymbol{U} = \boldsymbol{I}$.

with $\boldsymbol{S} \in \mathbb{R}^{r \times r}$ a diagonal matrix with the r nonzero singular values $\{\sigma_1, \dots, \sigma_r\}$ on the main diagonal. Instead of sending $\boldsymbol{x}$ directly, the transmitter computes and sends $\tilde{\boldsymbol{x}} = \boldsymbol{V}\boldsymbol{x}$. Once the receiver observes the channel output $\tilde{\boldsymbol{y}} = \boldsymbol{H}\tilde{\boldsymbol{x}} + \boldsymbol{n}$, it computes $\boldsymbol{y} = \boldsymbol{U}^{\mathsf{H}}\tilde{\boldsymbol{y}}$. As a result, the receiver obtains

$$\boldsymbol{y} = \boldsymbol{U}^{\mathsf{H}}\tilde{\boldsymbol{y}} = \boldsymbol{U}^{\mathsf{H}}(\boldsymbol{H}\tilde{\boldsymbol{x}} + \boldsymbol{n}) = \boldsymbol{U}^{\mathsf{H}}\boldsymbol{H}\boldsymbol{V}\boldsymbol{x} + \boldsymbol{U}^{\mathsf{H}}\boldsymbol{n} = \boldsymbol{U}^{\mathsf{H}}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{H}}\boldsymbol{V}\boldsymbol{x} + \boldsymbol{U}^{\mathsf{H}}\boldsymbol{n} = \boldsymbol{\Sigma}\boldsymbol{x} + \boldsymbol{U}^{\mathsf{H}}\boldsymbol{n}.$$

Let $\tilde{\boldsymbol{n}} = \boldsymbol{U}^{\mathsf{H}}\boldsymbol{n}$ be the transformed noise. Notice that because $\boldsymbol{U}$ is unitary, $\tilde{\boldsymbol{n}}$ and $\boldsymbol{n}$ have the same distribution,[2] i.e., the multiplication of $\tilde{\boldsymbol{y}}$ by $\boldsymbol{U}^{\mathsf{H}}$ has no effect on the perceived noise at the receiver. With $\boldsymbol{y} = \left(\boldsymbol{y}_1^{\mathsf{H}}, \boldsymbol{y}_2^{\mathsf{H}}\right)^{\mathsf{H}}$, $\boldsymbol{x} = \left(\boldsymbol{x}_1^{\mathsf{H}}, \boldsymbol{x}_2^{\mathsf{H}}\right)^{\mathsf{H}}$, and $\tilde{\boldsymbol{n}} = \left(\tilde{\boldsymbol{n}}_1^{\mathsf{H}}, \tilde{\boldsymbol{n}}_2^{\mathsf{H}}\right)^{\mathsf{H}}$, where $\boldsymbol{y}_1, \boldsymbol{x}_1, \tilde{\boldsymbol{n}}_1 \in \mathbb{C}^r$, we can see that $\boldsymbol{y}_1 = \boldsymbol{S}\boldsymbol{x}_1 + \tilde{\boldsymbol{n}}_1$, i.e., $y_i = \sigma_i x_i + \tilde{n}_i$, $\forall i \in [r]$. We thereby effectively turned the MIMO channel with N_t transmit antennas and N_r receive antennas into r independent single input single output channels with channel gains $\{\sigma_1, \dots, \sigma_r\}$ and noise levels equivalent to the initial noise at the receiver.

4.3.2 Application to Distributed Computations

Spatial diversity can help reduce the communication latency in distributed computing systems with wireless communication links. For example, by repeating tasks across multiple ENs, the ENs can pool their antennas and utilize the multiplexing gain to serve multiple users simultaneously. In order to compute the relevant element of $\tilde{\boldsymbol{x}}$, every participating EN needs access to the whole $\boldsymbol{x}$, which is the reason for replicating tasks across the ENs.

4.4 Shamir's Secret Sharing Scheme

Shamir's (n, k) secret sharing scheme (SSS) [43] encodes a secret σ from the finite field $\mathbb{F}_q$ of size q into n shares such that any $k \leq n$ shares can be used to recover σ while any $k-1$ shares or less do not reveal any information about σ. To properly understand how Shamir's SSS works, it is necessary to know the principle of one-time padding.

4.4.1 One-Time Padding

One-time padding is a mechanism to hide an $x \in \mathbb{F}_q$. In particular, let X be an RV over $\mathbb{F}_q$ with realization x and arbitrary probability mass function (PMF) $\Pr\{\mathsf{X} = x\}$ and let R be a uniformly distributed RV over $\mathbb{F}_q$ with PMF $\Pr\{\mathsf{R} = r\} = \frac{1}{q}$. We then have for $\mathsf{Y} = \mathsf{X} + \mathsf{R}$ that $\mathrm{I}(\mathsf{Y}; \mathsf{X}) = 0$, i.e., the mutual information between X and $\mathsf{X} + \mathsf{R}$ is zero.

To show this result, we first have to show that Y is uniformly distributed as well:

[2] Any RV with rotationally symmetric distribution is unaffected by a unitary transformation. Since AWGN is rotationally symmetric, $\tilde{\boldsymbol{n}}$ and $\boldsymbol{n}$ have the same distribution.

$$\begin{aligned}\Pr\{\mathsf{Y}=y\} &\overset{(a)}{=} \sum_x \Pr\{\mathsf{Y}=y|\mathsf{X}=x\}\cdot\Pr\{\mathsf{X}=x\}\\ &\overset{(b)}{=} \sum_x \Pr\{\mathsf{R}=y-x|\mathsf{X}=x\}\cdot\Pr\{\mathsf{X}=x\}\\ &\overset{(c)}{=} \sum_x \frac{1}{q}\cdot\Pr\{\mathsf{X}=x\}\\ &= \frac{1}{q}\cdot\sum_x \Pr\{\mathsf{X}=x\}\\ &= \frac{1}{q},\end{aligned}$$

where (a) follows from the law of total probability, (b) follows from $\mathsf{Y}=\mathsf{X}+\mathsf{R}$, and (c) follows from the independence of R and X and substituting $\Pr\{\mathsf{R}=r\}=\frac{1}{q}$.

Next, we can show that X and Y are conditionally independent, i.e., that the entropy about X is unaffected by observing Y:

$$\begin{aligned}\mathrm{H}(\mathsf{X},\mathsf{Y}) &= \mathrm{H}(\mathsf{X}|\mathsf{Y})+\mathrm{H}(\mathsf{Y}) = \mathrm{H}(\mathsf{Y}|\mathsf{X})+\mathrm{H}(\mathsf{X})\\ &\Rightarrow\\ \mathrm{H}(\mathsf{X}|\mathsf{Y}) &= \mathrm{H}(\mathsf{Y}|\mathsf{X})+\mathrm{H}(\mathsf{X})-\mathrm{H}(\mathsf{Y})\\ &= \mathrm{H}(\mathsf{X}+\mathsf{R}|\mathsf{X})+\mathrm{H}(\mathsf{X})-\mathrm{H}(\mathsf{Y})\\ &= \mathrm{H}(\mathsf{R})+\mathrm{H}(\mathsf{X})-\mathrm{H}(\mathsf{Y})\\ &\overset{(a)}{=} \mathrm{H}(\mathsf{X}),\end{aligned}$$

where (a) follows from $\mathrm{H}(\mathsf{R})=\mathrm{H}(\mathsf{Y})$ due to Y and R being identically distributed.

Lastly, it follows that the mutual information between X and Y is zero:

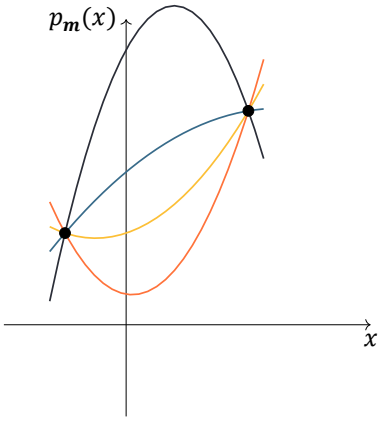
$$\mathrm{I}(\mathsf{Y};\mathsf{X}) = \mathrm{H}(\mathsf{X})-\mathrm{H}(\mathsf{X}|\mathsf{Y}) = 0.$$

As a result, Y does not reveal any information about X, i.e., knowing y does not help determine x. Also, R does not reveal any information about X because they are independent. However, knowing both y and r it is possible to recover x as $x=y-r$.

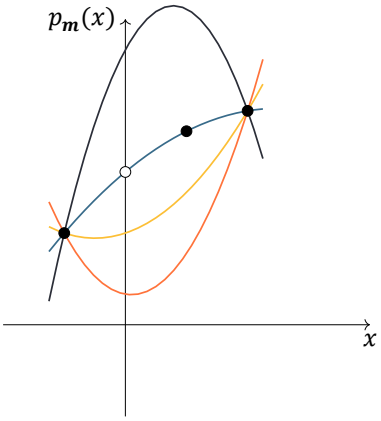
4.4.2 Shamir's Scheme

Shamir's SSS extends the idea of one-time padding to the more general case where knowing any k out of n realizations is sufficient to recover the secret σ whereas any $k-1$ or fewer realizations do not help in determining σ. In principle, the map $x \mapsto \{y,r\}$ from the one-time padding can be seen as a $(2,2)$ SSS with $\sigma=x$, albeit Shamir's SSS resulting in a different map, as described next.

In Shamir's (n,k) SSS, the secret Σ (with realization σ) is encoded together with $k-1$ independent and uniformly distributed (i.u.d.) RVs $\{\mathsf{R}_1,\mathsf{R}_2,\ldots,\mathsf{R}_{k-1}\}$



(a) Two evaluations do not suffice to determine $p_{\boldsymbol{m}}(x)$.



(b) Three evaluations suffice to determine $p_{\boldsymbol{m}}(x)$ and thereby the secret $\sigma = p_{\boldsymbol{m}}(0)$ (in white).

Figure 4.2: Example of Shamir's $(n, k = 3)$ SSS.

using a non-systematic Reed-Solomon code as described in Section 4.1 resulting in n shares $\{S_1, S_2, \dots, S_n\}$. In particular, we have

$$\boldsymbol{s} = \boldsymbol{m}\boldsymbol{G}, \tag{4.5}$$

with $\boldsymbol{s} = (s_1, s_2, \dots, s_n)$, $\boldsymbol{m} = (\sigma, r_1, r_2, \dots, r_{k-1})$, and $\boldsymbol{G}$ as in (4.3). (4.5) results in

$$s_i = \sigma + \sum_{j=1}^{k-1} r_j \cdot (\alpha^{i-1})^j. \tag{4.6}$$

Each share s_i is a padded version of the secret σ. It also becomes clear now why we abstain from evaluating $p_{\boldsymbol{m}}(x)$ in (4.1) at 0, i.e., why we focus on the case $n < q$ such that we can afford to evaluate $p_{\boldsymbol{m}}(x)$ at n distinct points without 0. If we would evaluate at 0, we would get $s_j = \sigma$ for some j, i.e., we would leak the secret σ.

In Fig. 4.2 we can see an example of an $(n, k = 3)$ scheme. In this case, $p_{\boldsymbol{m}}(x)$ is a degree two polynomial, i.e., a parabola. In Fig. 4.2(a) we have two evaluations of $p_{\boldsymbol{m}}(x)$ represented by the black points. We can see that there are many (q to be precise) polynomials that intersect these two points. Consequently, we are not able to uniquely determine either $p_{\boldsymbol{m}}(x)$ or $p_{\boldsymbol{m}}(0)$. In Fig. 4.2(b) we have a third evaluation which allows us to uniquely determine $p_{\boldsymbol{m}}(x)$ and thereby also the secret $\sigma = p_{\boldsymbol{m}}(0)$ which is represented by the white point.

It remains to show that for general n and k, any k shares are indeed sufficient to recover σ and that any $k-1$ shares do not leak any information about σ. More formally, for any $\mathcal{A} \subseteq \{S_1, \dots, S_n\}$ with $|\mathcal{A}| \geq k$ we have $H(\Sigma|\mathcal{A}) = 0$ and for any $\mathcal{B} \subseteq \{S_1, \dots, S_n\}$ with $|\mathcal{B}| < k$ we have $I(\Sigma; \mathcal{B}) = 0$. The first condition, $H(\Sigma|\mathcal{A}) = 0$, follows directly from the MDS property of RS codes. Any k error-free coded symbols are sufficient to recover the message $\boldsymbol{m}$, which includes the secret σ. That the second condition, $I(\Sigma; \mathcal{B}) = 0$, is met is not as obvious.

To reiterate the result from one-time padding, adding an i.u.d. RV R to an arbitrarily distributed secret Σ from a finite field masks the secret. It is straight forward to see that for two i.u.d. RVs R_1 and R_2 the set $\{\mathsf{Y}_1 = \Sigma + \mathsf{R}_1, \mathsf{Y}_2 = \Sigma + \mathsf{R}_2\}$ still masks Σ:

$$\begin{aligned} \mathrm{I}(\Sigma; \mathsf{Y}_1, \mathsf{Y}_2) &= \mathrm{I}(\Sigma; \mathsf{Y}_1) + \mathrm{I}(\Sigma; \mathsf{Y}_2 | \mathsf{Y}_1) \\ &\overset{(a)}{=} \mathrm{I}(\Sigma; \mathsf{Y}_1) + \mathrm{I}(\Sigma; \mathsf{Y}_2) \\ &\overset{(b)}{=} 0 + 0 = 0, \end{aligned}$$

where (a) follows from the fact that Y_1 is independent of both Σ (due to one-time padding) and Y_2 (as Y_2 is a one-time padded version of Y_1 with $\mathsf{Y}_2 = \mathsf{Y}_1 + \mathsf{R}'$ where $\mathsf{R}' = \mathsf{R}_2 - \mathsf{R}_1$ is uniformly distributed). Furthermore, (b) follows from one-time padding. This result is easily extended to more than two i.u.d. padded versions of Σ.

With (4.6) in mind, showing that $\mathcal{R} = \{\sum_{j=1}^{k-1} r_j \cdot (\alpha^{i-1})^j | i \in \mathcal{I} \subset [n], |\mathcal{I}| = k-1\}$ is a set of $k-1$ i.u.d. RVs is sufficient to prove that $\mathrm{I}(\Sigma; \mathcal{B}) = 0$. Suppose the elements in $\mathcal{R}$ are i.u.d. In that case, the shares in $\mathcal{B}$ are $k-1$ one-time padded versions of σ, and as shown previously (in detail for the case $k-1=2$), they do not reveal anything about Σ. With $\boldsymbol{r} = (r_1, r_2, \ldots, r_{k-1})$ we can write $\boldsymbol{r}'$, the vector with elements from $\mathcal{R}$, as $\boldsymbol{r}' = \boldsymbol{r}\boldsymbol{G}'$ where $\boldsymbol{G}'$ is a $(k-1)\times(k-1)$ submatrix of $\boldsymbol{G}$ in (4.3). Because $\boldsymbol{G}'$ is a submatrix of the Vandermonde matrix $\boldsymbol{G}$, all $k-1$ rows of $\boldsymbol{G}'$ are linearly independent, and as such, they form a basis for the $(k-1)$-dimensional vector space over $\mathbb{F}_q$. Consequently, there is a one-to-one mapping between $\boldsymbol{r}$ and $\boldsymbol{r}'$, and their distributions are identical. Hence, the elements in $\mathcal{R}$ are indeed $k-1$ i.u.d. RVs and we have proven that $\mathrm{I}(\Sigma; \mathcal{B}) = 0$.

4.4.3 Application to Distributed Computations

A master might not trust the workers with its private data. When it is reasonable to assume that not all workers collude, for example, when utilizing workers from different service providers, the master can apply secret sharing on its data before distributing it to prevent the workers from inferring information. The idea is similar to Section 4.1.2. The master can encode a matrix $\boldsymbol{A}$ together with random matrices using an (n, k) code as described above and distribute one coded matrix to each of the n workers. Consequently, any fewer than k workers cannot infer any information about $\boldsymbol{A}$, whereas the master will be able to recover the result after collecting sufficient local results.

4.5 Fixed-Point Arithmetic

While RS codes can be defined over general fields, one-time padding and hence Shamir's SSS require the existence of a uniform distribution over the used field. Consequently, Shamir's SSS is only defined over finite fields. However, many real-world problems are defined over the reals. As a result, we need a way to represent real numbers in a finite field. Naturally, representing an uncountably infinite

amount of numbers with a finite amount of field elements will result in only a limited range and precision of representable numbers.

Fixed-point numbers are an efficient way to represent an interval of real numbers as integers, which in turn are efficiently represented and computed on by computers. To this end, fixed-point numbers allocate a specific amount of bits for the fractional part of the real number. In particular, a fixed-point number $\tilde{x}$ of length ℓ bits with f bits reserved for the fractional part is given as $\tilde{x} = \bar{x} \cdot 2^{-f}$, where $\bar{x}$ is an integer from $\{-2^{\ell-1}, \dots, 2^{\ell-1} - 1\}$. Consequently, for any real number $x \in [-2^{\ell-f-1}, \dots, 2^{\ell-f-1} - 2^{-f}]$ there exists a fixed-point number $\tilde{x}$ of length ℓ and resolution f such that $|x - \tilde{x}| \leq 2^{-f-1}$. That means, for sufficiently large ℓ and f, we can represent any finite interval of real numbers by fixed-point numbers with a negligible error.

For a fixed resolution f, any fixed-point number $\tilde{x}$ is equally represented by its integer part $\bar{x}$. Consequently, we can conveniently store $\tilde{x}$ as an integer. What remains to be shown is that we can also perform additions and multiplications using the integer part of the fixed-point numbers. Let $\tilde{a}$ and $\tilde{b}$ be two fixed-point numbers with $\tilde{a} = \bar{a}2^{-f}$ and $\tilde{b} = \bar{b}2^{-f}$. For $\tilde{c} = \tilde{a} + \tilde{b}$, with $\tilde{c} = \bar{c}2^{-f}$, we have $\bar{c} = \bar{a}+\bar{b}$, i.e., addition of fixed-point numbers can be performed via simple integer addition. Note that in case $\bar{c} > 2^{\ell-1}$ we get an overflow, meaning that the result of $\tilde{a} + \tilde{b}$ can not be properly represented in the sense one might expect from adding two real numbers a and b. To avoid an undefined behavior, we wrap $\bar{c}$ around such that it is guaranteed to lie in the interval $\{-2^{\ell-1}, \dots, 2^{\ell-1} - 1\}$. However, this leads to maybe unexpected effects, such as the sum of two positive numbers possibly being negative.

For $\tilde{d} = \tilde{a} \cdot \tilde{b}$, with $\tilde{d} = \bar{d}2^{-f}$, we have $\bar{d} = \lfloor \bar{a} \cdot \bar{b} \cdot 2^{-f} \rfloor$, where $\bar{d}$ is again wrapped around in case an overflow occurs. In other words, multiplication of fixed-point numbers can be computed by an integer multiplication with subsequent scaling by 2^{-f} to retain the initial resolution of f bits.[3]

4.5.1 Application to Distributed Computations

We have seen that fixed-point numbers enable us to represent real numbers by integers and compute on real numbers through simple integer operations. Furthermore, for any prime number $q \geq 2^{\ell}$ the finite field $\mathbb{F}_q$ of size q can represent the integers in the set $\{-2^{\ell-1}, \dots, 2^{\ell-1} - 1\}$. In particular, the integers $0, 1, \dots, 2^{\ell-1}-1$ are represented by the field elements $0, 1, \dots, 2^{\ell-1}-1$, whereas integers $-1, -2, \dots, -2^{\ell-1}$ are represented by the field elements $q-1, q-2, \dots, q-2^{\ell-1}$. Furthermore, as long as there are no overflows beyond $2^{\ell-1} - 1$ or $-2^{\ell-1}$ during the integer operations, the integer operations can equally be performed in $\mathbb{F}_q$. This analogy means that we can represent an interval of real numbers with a finite field through fixed-point numbers. Consequently, we can apply the ideas from Section 4.4.3 to problems over the reals, albeit only with finite precision.

[3] Scaling, i.e., multiplying, a binary integer by a power of 2 can be performed via simple bit shift operations. For example, multiplying by 2^{-f} can be done by right-shifting all bits f positions.

Chapter 5

Paper Overview

In this chapter, I will summarize the two papers *"Privacy-Preserving Coded Mobile Edge Computing for Low-Latency Distributed Inference"* and *"CodedPaddedFL and CodedSecAgg: Straggler Mitigation and Secure Aggregation in Federated Learning"* that are part of this thesis.

5.1 Paper I

The paper *"Privacy-Preserving Coded Mobile Edge Computing for Low-Latency Distributed Inference"* applies ideas from coding theory, multiuser MIMO, and secret sharing to EC in order to mitigate the effect of straggling servers, reduce the communication load in the network, and guarantee user data privacy. The interplay of MDS codes for straggler mitigation and task replication to turn a single antenna network into a MIMO system was already studied in [20]. The novelty of Paper I lies in the tradeoff of task replication and secret sharing, two conflicting approaches. While replications help reduce the network's communication load, they are detrimental to the privacy guarantee of the underlying SSS.

For the desired privacy level, i.e., the number of ENs that are allowed to collude to infer private user information, the parameters of the proposed scheme are optimized to reduce the overall latency consisting of upload and download times to and from the edge servers, computation times at the edge servers, and decoding time at the devices. Furthermore, multiple scheme variations are proposed that further reduce the overall latency. An additional layer of coding at the edge server side helps when the computation time at the edge servers is comparatively high compared to the upload and download times because the resulting product code provides enhanced straggler mitigation capabilities. On the other hand, utilizing a priority queue in the download reduces the overall latency when communication times are high compared to the computation times.

Considering decoding in the overall latency, the proposed scheme, in fact, outperforms the scheme in [20] while providing user data privacy against a single EN.

5.2 Paper II

The paper *"CodedPaddedFL and CodedSecAgg: Straggler Mitigation and Secure Aggregation in Federated Learning"* proposes two schemes that utilize ideas from coding theory to mitigate the straggler effect in FL while retaining the privacy level of conventional FL through one-time padding and preventing model inversion attacks through secure aggregation via secret sharing, respectively. The core idea in both schemes is to introduce redundancy on the training data through codes, while CodedPaddedFL prevents information leakage from the sharing through one-time padding, and CodedSecAgg allows for secure aggregation.

The two schemes are tailored to linear regression; however, both schemes are applied to non-linear classification problems on the MNIST and Fashion-MNIST datasets through kernel embedding. For a considered scenario with 120 devices, CodedPaddedFL achieves a speedup factor of 18 compared to conventional FL to achieve 95% accuracy on the MNIST dataset and entails similar latency as the existing scheme [47] without a loss in user data privacy. For the same scenario, CodedSecAgg achieves a speedup factor of $6.6 - 18.7$ compared to LightSecAgg [35].

Chapter 6

Conclusion

This chapter concludes the first part of the thesis. The thesis introduced the general background of EC and FL, two emerging distributed computing frameworks. In particular, we looked at the characteristic problems both EC and FL face: the straggler effect, communication bottleneck, and privacy concerns. We then discussed linear regression, a typical application for both EC and FL. Next, possible approaches to mitigate the mentioned problems were explained in detail. Finally, the mitigation techniques were put in context in the form of a brief overview of the two papers *"Privacy-Preserving Coded Mobile Edge Computing for Low-Latency Distributed Inference"* and *"CodedPaddedFL and CodedSecAgg: Straggler Mitigation and Secure Aggregation in Federated Learning"* that are part of this thesis. For more details, the two papers can be found in Part B.

We saw that coding can help to significantly speed up both conventional EC and conventional FL while additionally providing user data privacy against several colluding adversaries.

Future Work

The presented straggler mitigation and privacy preservation strategies rely on the computations to be linear in the data. While linear operations are a prevalent part of various practical problems, many relevant non-linear problems exist. It is not straightforward to apply the schemes proposed in Paper I and Paper II to non-linear problems. Future work might include analysis and incorporation of existing approaches to adapt the proposed schemes to non-linear problems, e.g., through Lagrange encoding [15] or pice-wise linear approximations.

Furthermore, the proposed schemes are rigid in their construction. The parameters are derived from an expected number of stragglers in the network and the schemes do not adapt to the actual straggler situation at hand. Future work might investigate the application of rateless codes to the scenarios such as in [21, 39] or the possibility to utilize intermediate results from all non-stragglers as in [20].

Last but not least, the schemes in Paper II are tailored to federated stochastic gradient descent (FedSGD). A widely adopted alternative to FedSGD is federated averaging (FedAvg) where multiple local training epochs are performed prior to aggregating the local model updates at the central server. Future work might examine how to adapt the schemes in Paper II to FedAvg.

Bibliography

[1] J. Dean and S. Ghemawat, U.S. Patent 7,650,331, Jan., 2010.

[2] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2013.

[3] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing–a key technology towards 5G," *ETSI white paper*, no. 11, pp. 1–16, Sep. 2015.

[4] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artificial Intell. Stats. (AISTATS)*, Fort Lauderdale, FL, USA, Apr. 2017.

[5] H. Ludwig *et al.*, "IBM federated learning: An enterprise framework white paper v0. 1," *arXiv:2007.10987*, Jul. 2020.

[6] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[7] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Washington, DC, USA, Dec. 2016, pp. 1–6.

[8] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Proc. Neural Inf. Process. Syst. (NIPS)*, Long Beach, CA, USA, Dec. 2017, pp. 4403–4413.

[9] K. Lee, M. Lam, R. Pedersani, D. Papailiopoulos, and K. Ramachandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

[10] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Trans. Inf. Theory*, vol. 65, no. 7, pp. 4227–4242, Jul. 2019.

[11] S. Dutta, V. Cadambe, and P. Grover, ""Short-Dot": Computing large linear transforms distributedly using coded short dot products," *IEEE Trans. Inf. Theory*, vol. 65, no. 10, pp. 6171–6193, Oct. 2019.

[12] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Trans. Inf. Theory*, vol. 66, no. 1, pp. 278–301, Jan. 2020.

[13] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Sydney, Australia, Aug. 2017, pp. 3368–3376.

[14] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Proc. Neural Inf. Process. Syst. (NIPS)*, Long Beach, CA, USA, Dec. 2017, pp. 5440–5448.

[15] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. Int. Conf. Artificial Intell. Stats. (AISTATS)*, Okinawa, Japan, Apr. 2019, pp. 1215–1225.

[16] A. Severinson, A. Graell i Amat, E. Rosnes, F. Lázaro, and G. Liva, "A droplet approach based on Raptor codes for distributed computing with straggling servers," in *Proc. 10th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Hong Kong, China, Dec. 2018.

[17] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, Dec. 2019.

[18] K. Li, M. Tao, and Z. Chen, "Exploiting computation replication for mobile edge computing: A fundamental computation-communication tradeoff study," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4563–4578, Jul. 2020.

[19] ——, "A computation-communication tradeoff study for mobile edge computing networks," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Paris, France, Jul. 2019, pp. 2639–2643.

[20] J. Zhang and O. Simeone, "On model coding for distributed inference and transmission in mobile edge computing systems," *IEEE Commun. Lett.*, vol. 23, no. 6, pp. 1065–1068, Jun. 2019.

[21] K. Li, M. Tao, J. Zhang, and O. Simeone, "Multi-cell mobile edge coded computing: Trading communication and computing for distributed matrix multiplication," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Los Angeles, CA, USA, Jun. 2020, pp. 215–220.

[22] K. Li, M. Tao, J. Zhang, and O. Simeone, "Coded computing and cooperative transmission for wireless distributed matrix multiplication," *IEEE Trans. Commun.*, vol. 69, no. 4, pp. 2224–2239, Apr. 2021.

[23] A. Frigård, S. Kumar, E. Rosnes, and A. Graell i Amat, "Low-latency distributed inference at the network edge using rateless codes," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, Berlin, Germany, Sep. 2021.

[24] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure coded computing using secret sharing via staircase codes," *IEEE Trans. Commun.*, vol. 68, no. 8, pp. 4609–4619, Aug. 2020.

[25] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. El Rouayheb, and H. Seferoglu, "Prac: Private and rateless adaptive coded computation at the edge," in *Proc. SPIE Defense + Commercial Sensing*, Baltimore, MD, USA, May 2019.

[26] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. Int. Conf. Artificial Intell. Stats. (AISTATS)*, Naha, Japan, Apr. 2019, pp. 1215–1225.

[27] H. Yang and J. Lee, "Secure distributed computing with straggling servers using polynomial codes," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 1, pp. 141–150, Jan. 2019.

[28] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Denver, CO, USA, Oct. 2015, pp. 1322–1333.

[29] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *Proc. IEEE Int. Conf. Comp. Commun. (INFOCOM)*, Paris, France, Apr./May 2019, pp. 2512–2520.

[30] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Dallas, TX, USA, Oct./Nov. 2017, pp. 1175–1191.

[31] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramachandran, "FastSecAgg: Scalable secure aggregation for privacy-preserving federated learning," in *Int. Workshop Fed. Learn. User Privacy Data Confidentiality*, Vienna, Austria, Jul. 2020.

[32] J. So, B. Güler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 1, pp. 479–489, Mar. 2021.

[33] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly)logarithmic overhead," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, online, Nov. 2020, pp. 1253–1269.

[34] A. R. Elkordy and A. S. Avestimehr, "HeteroSAg: Secure aggregation with heterogeneous quantization in federated learning," *IEEE Trans. Commun.*, vol. 70, no. 4, pp. 2372–2386, Apr. 2022.

[35] J. So, C. He, C.-S. Yang, S. Li, Q. Yu, R. E. Ali, B. Güler, and S. Avestimehr, "LightSecAgg: a lightweight and versatile design for secure aggregation in federated learning," in *Proc. Mach. Learn. Syst. (MLSys)*, Santa Clara, CA, USA, Aug./Sep. 2022.

[36] Y. Zhao and H. Sun, "Information theoretic secure aggregation with user dropouts," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Melbourne, Australia, Jul. 2021, pp. 1124–1129.

[37] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and verifiable federated learning," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 911–926, 2020.

[38] T. Jahani-Nezhad, M. A. Maddah-Ali, S. Li, and G. Caire, "SwiftAgg: Communication-efficient and dropout-resistant secure aggregation for federated learning with worst-case security guarantees," Feb. 2022, arXiv:2202.04169.

[39] A. R. Chowdhury, C. Guo, S. Jha, and L. van der Maaten, "EIFFeL: Ensuring integrity for federated learning," Dec. 2021, arXiv:2112.12727.

[40] T. Jahani-Nezhad, M. A. Maddah-Ali, S. Li, and G. Caire, "SwiftAgg+: Achieving asymptotically optimal communication load in secure aggregation for federated learning," Mar. 2022, arXiv:2203.13060.

[41] J. So, R. E. Ali, B. Güler, and A. S. Avestimehr, "Secure aggregation for buffered asynchronous federated learning," in *1st NeurIPS Workshop New Frontiers Fed. Learn. (NFFL)*, online, Dec. 2021.

[42] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," in *Proc. Int. Conf. Artificial Intell. Stats. (AISTATS)*, online, Mar. 2022, pp. 3581–3607.

[43] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[44] A. Rényi, "On the theory of order statistics," *Acta Mathematica Academiae Scientiarum Hungaricae*, vol. 4, nos. 3–4, pp. 191–231, 1953.

[45] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Proc. Neural Inf. Process. Syst. (NIPS)*, Vancouver, BC, Canada, Dec. 2007, pp. 1177–1184.

[46] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, 1960.

[47] S. Prakash, S. Dhakal, M. R. Akdeniz, Y. Yona, S. Talwar, S. Avestimehr, and N. Himayat, "Coded computing for low-latency federated learning over wireless edge networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 233–250, Jan. 2021.

Part B

Papers

Paper I

Privacy-Preserving Coded Mobile Edge Computing for Low-Latency Distributed Inference

Reent Schlegel, Siddhartha Kumar, Eirik Rosnes, and Alexandre Graell i Amat

IEEE Journal on Selected Areas in Communications, vol. 40, no. 3, pp. 788-799, March 2022.

Parts of this paper were presented at the IEEE Global Communications Conference (GLOBECOM), Taipei, Taiwan, December 2020.

The layout has been revised.

Abstract

We consider a mobile edge computing scenario where a number of devices want to perform a linear inference $\boldsymbol{W}\boldsymbol{x}$ on some local data $\boldsymbol{x}$ given a network-side matrix $\boldsymbol{W}$. The computation is performed at the network edge over a number of edge servers. We propose a coding scheme that provides information-theoretic privacy against z colluding (honest-but-curious) edge servers, while minimizing the overall latency—comprising upload, computation, download, and decoding latency—in the presence of straggling servers. The proposed scheme exploits Shamir's secret sharing to yield data privacy and straggler mitigation, combined with replication to provide spatial diversity for the download. We also propose two variants of the scheme that further reduce latency. For a considered scenario with 9 edge servers, the proposed scheme reduces the latency by 8% compared to the nonprivate scheme recently introduced by Zhang and Simeone, while providing privacy against an honest-but-curious edge server.

1 Introduction

Mobile edge computing is a key enabler of delay-critical internet-of-things applications that rely on large data computing services [1], and has become a pillar of the 5G mobile network [2]. Offloading computations to far-away cloud services can be infeasible due to bandwidth constraints on the backhaul network and possibly large communication latency [1]. To circumvent these shortcomings, the edge computing paradigm moves the computation power closer to the devices generating the data.

Distributing computations over a number of servers at the edge of the wireless network leads to major challenges, among them the presence of *straggling* servers—the computation latency is dominated by the slowest server. The straggler problem has been addressed in the neighboring field of distributed computing in data centers by means of coding [3–13]. The key idea in distributed computing is to introduce redundant computations across servers via an erasure correcting code such that the results from a subset of the servers is sufficient to recover (decode) the desired computation. Hence, the latency is no longer dominated by the slowest servers. Maximum distance separable (MDS) codes have been shown to provide excellent straggler resiliency [3, 4]. Most works on coded computing neglect the impact of the decoding complexity on the latency. An exception is [5, 6], where it was shown that the decoding latency may severely impact the overall latency. Long MDS codes, in particular, entail a high decoding complexity, which may impair the overall latency.

In edge computing, besides the straggler problem, the incurred latency of uploading and downloading data through the wireless links is a genuine problem. To reduce the communication latency, in [14, 15] subtasks are replicated across

edge servers to introduce spatial diversity such that the edge servers can utilize zero-forcing precoding and serve multiple users simultaneously. More recently, the authors in [16–18] proposed to combine subtask replication for spatial diversity with an MDS code for straggler mitigation, borrowing the coding ideas from distributed computing. These works, however, neglect the latency entailed by the decoding operation. In [39], a scheme combining rateless codes with irregular repetition was proposed, yielding significantly lower latency (comprising the decoding latency) than the scheme in [16–18].

Performing computations over possibly untrustworthy edge servers raises also privacy concerns. The problem of user data privacy in the context of distributed computing in data centers in the presence of stragglers has been addressed in, e.g., [20–23]. The underlying idea in these works is to utilize some form of secret sharing, i.e., encode the confidential user data together with random data such that small subsets of servers do not gain information about the confidential data. we consider a similar scenario to the one in [17, 39] where multiple users wish to perform a linear inference $\boldsymbol{W}\boldsymbol{x}$ on some local data $\boldsymbol{x}$ given a network-side matrix $\boldsymbol{W}$. Such operations arise in, e.g., recommender systems based on collaborative filtering, like a shopping center application providing product recommendations and corresponding price offers[24]. Each customer has its preferences, which are encoded by an attribute vector $\boldsymbol{x}$. Based on a customer's preferences, the application recommends products by mapping from a customer's preference (vector $\boldsymbol{x}$) to the likelihood that he/she would enjoy a given product via the system matrix $\boldsymbol{W}$. For this scenario, we present a coding scheme that guarantees information-theoretic user data privacy against z compromised (honest-but-curious) edge servers that collaborate to infer users' data, while minimizing the incurred overall latency—comprising upload, computation, download, and decoding latency. The proposed scheme is based on Shamir's secret sharing (SSS) scheme [47] to achieve data privacy as well as straggler mitigation—thereby reducing computation latency—combined with replication of subtasks across multiple edge servers to allow for spatial diversity and joint beamforming (by means of zero-forcing precoding) in the download to reduce communication latency. A key feature of the proposed scheme is that, unlike the existing (nonprivate) schemes for straggler mitigation in edge computing [16–18], redundancy is introduced on the users' data—which enables privacy—instead of on the network-side matrix $\boldsymbol{W}$.

We also introduce two variants of the scheme that further reduce latency. First, we note that the download phase can be performed simultaneously to the computation phase once the upload phase is completed, which reduces the overall latency especially when the communication cost is relatively high compared to the computation cost. To exploit this, we introduce a priority queue to determine the order in which partial results should be downloaded from the edge servers. Second, we introduce an additional level of coding on $\boldsymbol{W}$. We show that the combination of the SSS code and the code on $\boldsymbol{W}$ results in a product code over intermediate results. Decoding can then be performed iteratively, iterating between the row and column component decoders.

The proposed scheme entails an inherent tradeoff between computation latency due to straggling servers, communication latency, and user data privacy. Interestingly, for a considered scenario with 9 edge servers, the proposed scheme

reduces the latency by 8% compared to the nonprivate scheme in [17], while providing privacy against a single edge server. This somewhat surprising result is explained by the high decoding complexity of the scheme in [17] due to the use of a long MDS code (on $\boldsymbol{W}$), while the proposed schemes rely on short codes over both users' data and $\boldsymbol{W}$. Higher privacy levels can be achieved at the expense of higher latency. Furthermore, the additional coding on $\boldsymbol{W}$ significantly reduces the variance of the latency, which, for a scenario where the linear inference needs to be performed within a deadline, increases the probability of meeting the deadline.

Notation: Vectors and matrices are written in lowercase and uppercase bold letters, respectively, e.g., $\boldsymbol{a}$ and $\boldsymbol{A}$, and all vectors are represented as column vectors. The transpose of vectors and matrices is denoted by $(\cdot)^\mathsf{T}$. $\mathrm{GF}(q)$ denotes the finite field of order q and $\mathbb{N}$ denotes the positive integers. We use the notation $[a]$ to represent the set of integers $\{1, 2, \ldots, a\}$. Furthermore, $\lceil a/b \rceil$ is the smallest integer larger than or equal to a/b and $\lfloor a/b \rfloor$ is the largest integer smaller than or equal to a/b. We represent permutations in cycle notation, e.g., the permutation $\pi = (1\ 3\ 2\ 4)$ maps $1 \mapsto 3$, $3 \mapsto 2$, $2 \mapsto 4$, and $4 \mapsto 1$. In addition, $\pi(i)$ is the image of i under π, e.g., $\pi(1) = 3$. Applying π recursively i times is denoted by π^i, e.g., $\pi^2(3) = 4$, and π^0 is an identity, e.g., $\pi^0(2) = 2$. The expected value of a random variable X is denoted by $\mathbb{E}[X]$.

2 System Model

We consider a scenario with u single-antenna users, $\mathsf{u}_1, \ldots, \mathsf{u}_u$, each wanting to compute the linear inference operation $\boldsymbol{y}_i = \boldsymbol{W}\boldsymbol{x}_i$ on its local and private data $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,r})^\mathsf{T} \in \mathrm{GF}(q)^r$ for some network-side public matrix $\boldsymbol{W} \in \mathrm{GF}(q)^{m \times r}$. The operation is offloaded to the edge and is performed in a distributed fashion over a number of edge servers—hereafter referred to as edge nodes (ENs). We assume that there are $e_{\max}$ ENs available at the network edge, and that the linear inference is performed over $e \leq e_{\max}$ ENs, where e can be optimized. The e ENs that perform the computation tasks are denoted $\mathsf{e}_1, \ldots, \mathsf{e}_e$. Each EN has a storage capacity μ, $0 < \mu \leq 1$, which is the fraction of $\boldsymbol{W}$ each EN can store, i.e., each EN can store up to μmr elements from $\mathrm{GF}(q)$. We assume that $\boldsymbol{W}$ stays constant over a sufficient amount of time so that it can be stored on the ENs offline. The system model is depicted in Fig. I.1.

2.1 Computation Runtime Model

The ENs are in general multi-task nodes, may run several applications in parallel, and need to serve many users. As a result, they may straggle. We model this behavior with a random setup time λ_j for each EN e_j. The setup time is the time it takes an EN to start the computation after it received all the necessary data. Here, we assume the widely-adopted model in which the setup times are independent and identically distributed (i.i.d.) and modeled by an exponential distribution with parameter η, such that $\mathbb{E}[\lambda_j] = 1/\eta$ [13, 17, 26]. Once set up, an EN needs τ time units to compute an inner product in $\mathrm{GF}(q)^r$ for each of the users, i.e., it takes an EN τ time units to do r multiplications and $r - 1$ additions for all users.

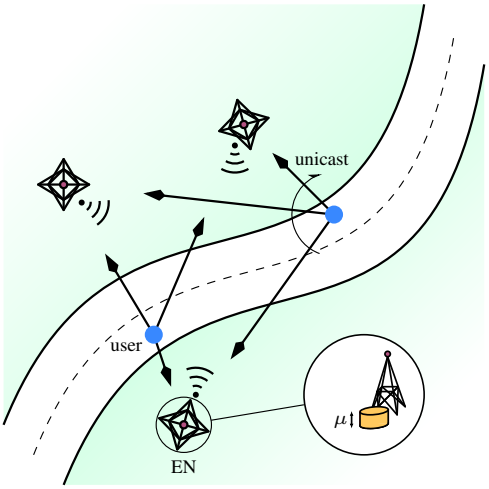


Figure I.1: A mobile edge computing network with two users and three ENs.

Consequently, to compute d inner products for each user (ud inner products in total), EN e_j incurs a latency of

$$\lambda_j + d\tau\,.$$

We define the *normalized computation* latency of EN e_j (normalized by τ) as

$$\mathsf{L}_j^{\mathsf{comp}} = \frac{\lambda_j}{\tau} + d\,.$$

The ENs have superior computing capabilities compared to the users. In particular, we assume that the users need δ normalized time units to perform r multiplications and $r-1$ additions.

2.2 Communication

The users have to upload their data to the ENs as well as download the results of the computations from the ENs. We denote by γ the normalized time it takes for both upload and download to unicast a symbol $\alpha \in \mathrm{GF}(q)^u$ (i.e., an element from $\mathrm{GF}(q)$ for each user). Consequently, the normalized time incurred by all users uploading their data (i.e., u vectors in $\mathrm{GF}(q)^r$) to a single EN is

$$\mathsf{L}^{\mathsf{comm,up}} = \gamma r\,.$$

For the upload of the private data $\{\boldsymbol{x}_i\}$ from the u users to the e ENs, we assume that transmission occurs sequentially, i.e., we consider time-division multiple access, whereas in the download the ENs can transmit simultaneously to multiple users by utilizing joint beamforming based on zero-forcing precoding [14–17, 27, 28]. More precisely, a symbol available at ρ ENs can be transmitted simultaneously to $\min\{\rho, u\}$ users with a normalized communication latency of

$\gamma/\min\{\rho, u\}$ in the high signal-to-noise (SNR) regime. The normalized communication latency in the download (in the high SNR regime) incurred by transmitting v symbols $\alpha_1, \ldots, \alpha_v$ in $\mathrm{GF}(q)^u$, one element from $\mathrm{GF}(q)$ for each user, where symbol α_i is available at ρ_i ENs, is

$$\mathsf{L}^{\mathsf{comm,down}} = \gamma \sum_{i=1}^{v} \frac{1}{\min\{\rho_i, u\}}.$$

The communication latency is then

$$\mathsf{L}^{\mathsf{comm}} = \mathsf{L}^{\mathsf{comm,up}} + \mathsf{L}^{\mathsf{comm,down}}.$$

2.3 Privacy and Problem Formulation

The ENs may not be trustworthy or may be compromised. Further, the compromised ENs may collaborate to infer the data of the users. In this paper, we assume that up to z ENs may be compromised and may collude. Our goal is to perform the inference problem over e ENs privately (so that the compromised ENs gain no information in an information-theoretic sense about the private data) while minimizing the overall latency,

$$\mathsf{L} = \mathsf{L}^{\mathsf{comp}} + \mathsf{L}^{\mathsf{comm}} + \mathsf{L}^{\mathsf{dec}},$$

encompassing the computation and communication latency, as well as the latency incurred by the decoding operation, denoted by $\mathsf{L}^{\mathsf{dec}}$ and discussed in Section 4.3.

3 Private Distributed Linear Inference

In this section, we present a privacy-preserving coded scheme that allows u users to perform the linear inference $\{\boldsymbol{W}\boldsymbol{x}_i\}$ over e ENs without revealing any information to any subset of z colluding ENs. A distinguishing feature of the proposed scheme is that, unlike the (nonprivate) scheme in [17], which yields straggler mitigation by introducing redundancy on matrix $\boldsymbol{W}$, it introduces redundancy on the users' data, by means of secret sharing according to [47], which allows to achieve straggler mitigation while guaranteeing user data privacy simultaneously.

3.1 Secret Sharing

We consider the SSS scheme to yield privacy. An (n, k) SSS scheme divides a secret into n pieces, referred to as *shares*, such that any k or more shares are sufficient to recover the data, while less than k shares do not reveal any information about data.

The proposed scheme is as follows. Each user u_i uses an (n, k) SSS scheme to compute n shares of its private data $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,r})^\top$. In particular, user u_i encodes each data entry $x_{i,l}$ along with $k-1$ i.i.d. uniform random symbols $r_{i,l}^{(1)}, \ldots, r_{i,l}^{(k-1)}$ from $\mathrm{GF}(q)$ using a nonsystematic (n, k) Reed-Solomon (RS) code

over GF(q) to obtain n coded symbols $s_{i,l}^{(1)}, \dots, s_{i,l}^{(n)}$. Let $\{\boldsymbol{r}_i^{(\kappa)} = (r_{i,1}^{(\kappa)}, \dots, r_{i,r}^{(\kappa)})^\top \mid \kappa \in [k-1]\}$ be the set of vectors of uniform random symbols used by user u_i. For each $h \in [n]$, the h-th share of user u_i is

$$\boldsymbol{s}_i^{(h)} = \left(s_{i,1}^{(h)}, \dots, s_{i,r}^{(h)}\right)^\top.$$

We collect the h-th share of all users in the matrix

$$\boldsymbol{S}^{(h)} = \left(\boldsymbol{s}_1^{(h)}, \boldsymbol{s}_2^{(h)}, \dots, \boldsymbol{s}_u^{(h)}\right) \in \mathrm{GF}(q)^{r \times u}. \tag{I.1}$$

The following theorem proves that the linear inference operations $\{\boldsymbol{y}_i = \boldsymbol{W}\boldsymbol{x}_i\}$ can be computed for all users from a given set of computations based on the matrices of shares $\boldsymbol{S}^{(1)}, \dots, \boldsymbol{S}^{(n)}$, while providing privacy against up to $k-1$ colluding ENs—which collectively have access to up to $k-1$ distinct matrices of shares.

Theorem 1. *Consider u users with their respective private data $\boldsymbol{x}_i \in \mathrm{GF}(q)^r$, $i \in [u]$. Use an (n, k) SSS scheme on each $\boldsymbol{x}_i$ to obtain the matrices of shares $\boldsymbol{S}^{(1)}, \dots, \boldsymbol{S}^{(n)}$ in* (I.1)*. Let $\boldsymbol{W} \in \mathrm{GF}(q)^{m \times r}$ be a public matrix and $\mathcal{I} \subseteq [n]$ a set of indices with cardinality $|\mathcal{I}| = k$. Then, the set of computations $\{\boldsymbol{W}\boldsymbol{S}^{(h)} \mid h \in \mathcal{I}\}$ allows to recover the computations $\{\boldsymbol{W}\boldsymbol{x}_i\}$ of all users. Moreover, for any set $\mathcal{J} \subseteq [n]$ with $|\mathcal{J}| < k$, $\{\boldsymbol{W}\boldsymbol{S}^{(h)} \mid h \in \mathcal{J}\}$ reveals no information about $\{\boldsymbol{W}\boldsymbol{x}_i\}$.*

The proof is given in Appendix A. The following corollary gives a sufficient condition to recover the private computations $\{\boldsymbol{W}\boldsymbol{x}_i\}$.

Corollary 1 (Sufficient recovery condition)**.** *Consider an edge computing scenario where the public matrix $\boldsymbol{W}$ is partitioned row-wise into b disjoint submatrices $\boldsymbol{W}_\ell \in \mathrm{GF}(q)^{\frac{m}{b} \times r}$, $\ell \in [b]$, and the private data is $\{\boldsymbol{x}_i\}$. Then, the private computations $\{\boldsymbol{W}\boldsymbol{x}_i\}$ can be recovered from the computations in the sets*

$$\mathcal{S}_\ell \triangleq \{\boldsymbol{W}_\ell \boldsymbol{S}^{(h)} \mid h \in \mathcal{I}\},\ \ell \in [b], \tag{I.2}$$

for any fixed set $\mathcal{I} \subseteq [n]$ with cardinality $|\mathcal{I}| = k$.

Proof. From Theorem 1, for a given $\ell \in [b]$, the computations in the set $\{\boldsymbol{W}_\ell \boldsymbol{x}_i\}$ can be recovered from the computations in the set $\mathcal{S}_\ell$. Then, we obtain

$$\boldsymbol{W}\boldsymbol{x}_i = \left((\boldsymbol{W}_1\boldsymbol{x}_i)^\top, (\boldsymbol{W}_2\boldsymbol{x}_i)^\top, \dots, (\boldsymbol{W}_b\boldsymbol{x}_i)^\top\right)^\top,\ \forall i \in [u].$$

□

Given the SSS scheme, the proposed scheme can be reduced to two combinatorial problems: the assignment of submatrices $\{\boldsymbol{W}_\ell\}$ to the ENs such that no EN stores more than a fraction μ of $\boldsymbol{W}$, and the assignment of matrices of shares $\{\boldsymbol{S}^{(h)}\}$ to the ENs such that no z colluding ENs gain any information about the data $\{\boldsymbol{x}_i\}$. We require that the combination of the assignments guarantees the users to obtain the compuations in (I.2), such that the users have access to sufficient data to recover $\{\boldsymbol{W}\boldsymbol{x}_i\}$. In the following two subsections, we describe the assignment of $\{\boldsymbol{W}_\ell\}$ and $\{\boldsymbol{S}^{(h)}\}$ to the ENs.

3.2 Assignment of W to the Edge Nodes

To create joint beamforming opportunities in the download, we allow for replications of the same $\boldsymbol{W}_\ell$ across different ENs. Submatrices are assigned to the ENs as follows. In order to satisfy the storage constraint, i.e., no EN can store more than a fraction μ of $\boldsymbol{W}$, we select $p \in \mathbb{N}$ such that $p/e \leq \mu$ and partition $\boldsymbol{W}$ row-wise into e submatrices as

$$\boldsymbol{W} = \left(\boldsymbol{W}_1^\top, \boldsymbol{W}_2^\top, \dots, \boldsymbol{W}_e^\top\right)^\top.$$

We then assign p submatrices to each of the e ENs. To this scope, we define a matrix of indices $\boldsymbol{I}_{\mathrm{w}}$, of dimensions $p \times e$, which prescribes the assignment of submatrices to the ENs. The assignment has the following combinatorial structure. Consider a cyclic permutation group of order e with generator π. We construct $\boldsymbol{I}_{\mathrm{w}}$ as

$$\boldsymbol{I}_{\mathrm{w}} = \begin{pmatrix} \pi^0(1) & \pi^0(2) & \cdots & \pi^0(e) \\ \pi^1(1) & \pi^1(2) & \cdots & \pi^1(e) \\ \vdots & \vdots & \ddots & \vdots \\ \pi^{p-1}(1) & \pi^{p-1}(2) & \cdots & \pi^{p-1}(e) \end{pmatrix} \tag{I.3}$$

and define the set of indices

$$\mathcal{I}_j^{\mathrm{w}} = \{\pi^0(j), \dots, \pi^{p-1}(j)\} \tag{I.4}$$

for $j \in [e]$ as the set containing the entries in column j of $\boldsymbol{I}_{\mathrm{w}}$. Then, we assign the submatrices $\{\boldsymbol{W}_\ell \mid \ell \in \mathcal{I}_j^{\mathrm{w}}\}$ to EN e_j, i.e., e_j is assigned the submatrices $\{\boldsymbol{W}_\ell\}$ with indices ℓ in the j-th column of $\boldsymbol{I}_{\mathrm{w}}$. For example, if $\pi = (1\ e\ e-1\ \cdots\ 2)$, we have

$$\boldsymbol{I}_{\mathrm{w}} = \begin{pmatrix} 1 & 2 & \cdots & e \\ e & 1 & \cdots & e-1 \\ \vdots & \vdots & \ddots & \vdots \\ e-p+2 & e-p+3 & \cdots & e-p+1 \end{pmatrix},$$

and EN e_2 stores $\boldsymbol{W}_2, \boldsymbol{W}_1, \boldsymbol{W}_e, \dots, \boldsymbol{W}_{e-p+3}$.

This assignment of submatrices to ENs bears some resemblance with fractional repetition (FR) codes [29]. FR codes were proposed in the context of distributed storage systems and yield the property that any ζ storage nodes have access to at least ψ distinct symbols/packets of a ψ-dimensional MDS code such that users can recover the data by decoding the MDS code after contacting ζ storage nodes. By guaranteeing that all pairs of storage nodes share exactly θ packets (utilizing Steiner systems such as the Fano plane), the authors can derive lower bounds on the number of distinct packets across ζ storage nodes. From this lower bound, the above-mentioned property (i.e., that any ζ storage nodes have access to at least ψ distinct symbols/packets) follows. In contrast, our goal is to achieve significant replication of submatrices $\boldsymbol{W}_\ell$ at the ENs, which we achieve by a cyclic structure. We do not have the requirement that any two ENs share exactly θ packets. Furthermore, one of our proposed schemes (introduced in Section 5.2) allows for irregular repetition of packets across ENs, while an essential requirement of FR codes is that packets are repeated the same amount of times across nodes. To summarize,

both our assignment of submatrices and FR codes are combinatorial designs, but serve different purposes. Notably, our assignment is much less structured than FR codes.

The ENs process the assigned submatrices of $\boldsymbol{W}$ in the same order as their indices appear in the rows of $\boldsymbol{I}_{\mathsf{w}}$. We define $\phi_j^{\mathsf{w}}(\ell')$ for $\ell' \in [p]$ to be the map from ℓ' to the index of the ℓ'-th assigned submatrix of EN e_j.

3.3 Assignment of Shares to the Edge Nodes

On the basis of the assignment of the submatrices of $\boldsymbol{W}$, to guarantee privacy, we now have to define the assignment of matrices of shares such that no z colluding ENs have access to k or more distinct matrices of shares, while the users should be guaranteed to obtain the computations in (I.2). Here, we restrict the number of shares n to be at most equal to the number of ENs, i.e., we require $n \leq e$. As with the submatrices of $\boldsymbol{W}$, we allow replicating shares across ENs to exploit joint beamforming opportunities in the download. However, this may lead to multiple shares being assigned to a single EN, which presents difficulties in the design of a private scheme, because having multiple shares available at a single EN results in a privacy level z lower than that of the SSS scheme (k). For example, if all ENs have access to two matrices of shares, the scheme only provides privacy against any $z = \lfloor (k-1)/2 \rfloor$ colluding ENs.

Alike to $\boldsymbol{I}_{\mathsf{w}}$, let $\boldsymbol{I}_{\mathsf{s}}$ be the index matrix that prescribes the assignment of matrices of shares to the ENs—the users upload their shares to the ENs according to $\boldsymbol{I}_{\mathsf{s}}$. The assignment has the following structure. Given the generator π used to assign the submatrices of $\boldsymbol{W}$ to the ENs, we construct the $(\beta+1) \times e$ index matrix $\boldsymbol{I}_{\mathsf{s}}$ as

$$\boldsymbol{I}_{\mathsf{s}} = \begin{pmatrix} \pi^0(1) & \pi^0(2) & \cdots & \pi^0(e) \\ \pi^{e-p}(1) & \pi^{e-p}(2) & \cdots & \pi^{e-p}(e) \\ \vdots & \vdots & \ddots & \vdots \\ \pi^{\beta(e-p)}(1) & \pi^{\beta(e-p)}(2) & \cdots & \pi^{\beta(e-p)}(e) \end{pmatrix}, \tag{I.5}$$

where $\beta = \lceil e/p \rceil - 1$. Define the set of indices

$$\mathcal{J}_j^{\mathsf{s}} = \{\pi^0(j), \ldots, \pi^{\beta(e-p)}(j)\} \backslash \{n+1, n+2, \ldots, e\} \tag{I.6}$$

as the subset of entries in column j of $\boldsymbol{I}_{\mathsf{s}}$ that are in $[n]$. We have

$$|\mathcal{J}_j^{\mathsf{s}}| = \lceil \lceil e/p \rceil \cdot n/e \rceil \triangleq a\,, \tag{I.7}$$

as we keep only a fraction $\lceil n/e \rceil$ of the shares corresponding to the $\beta + 1 = \lceil e/p \rceil$ used permutations in $\boldsymbol{I}_{\mathsf{s}}$. Then, user u_i transmits the shares $\{\boldsymbol{s}_i^{(h)} \mid h \in \mathcal{J}_j^{\mathsf{s}}\}$ to EN e_j, i.e., EN e_j is assigned a matrices of shares $\{\boldsymbol{S}^{(h)}\}$ with indices h in the j-th column of $\boldsymbol{I}_{\mathsf{s}}$ that are in $[n]$. Consequently, z colluding ENs have access to az possibly distinct matrices of shares. To guarantee user data privacy against any subset of z colluding ENs, we have to impose the constraint $k \geq az + 1$.

Similar to the submatrices of $\boldsymbol{W}$, the shares are processed by the ENs in the same order as their indices appear in the rows of $\boldsymbol{I}_{\mathsf{s}}$. We define $\phi_j^{\mathsf{s}}(h')$ for $h' \in [a]$ to be

the map from h' to the index of the h'-th assigned matrix of shares of EN e_j. For all $\ell' \in [p]$, EN e_j computes $\boldsymbol{W}_{\phi_j^{\mathsf{w}}(\ell')}\boldsymbol{S}^{(\phi_j^{\mathsf{s}}(h'))}$ before moving on to the next matrix of shares $\boldsymbol{S}^{(\phi_j^{\mathsf{s}}(h'+1))}$. The following theorem shows that the combined assignment of submatrices and shares to the ENs allow each user u_i to obtain its desired result $\boldsymbol{W}\boldsymbol{x}_i$ while preserving privacy against up to z colluding ENs.

Theorem 2. *Consider an edge computing network consisting of u users and e ENs, each with a storage capacity corresponding to a fraction μ, $0 < \mu \leq 1$, of $\boldsymbol{W}$, and an $(n, k \geq az + 1)$ SSS scheme, with $n \leq e$ and a given in* (I.7)*. For $j \in [e]$, EN e_j stores the submatrices of $\boldsymbol{W}$ from the set $\{\boldsymbol{W}_\ell \mid \ell \in \mathcal{I}_j^{\mathsf{w}}\}$ with $\mathcal{I}_j^{\mathsf{w}}$ defined in* (I.4)*. Furthermore, it receives the matrices of shares from the set $\{\boldsymbol{S}^{(h)} \mid h \in \mathcal{I}_j^{\mathsf{s}}\}$ with $\mathcal{I}_j^{\mathsf{s}}$ defined in* (I.6)*, and computes and returns the set $\{\boldsymbol{W}_\ell\boldsymbol{S}^{(h)} \mid \ell \in \mathcal{I}_j^{\mathsf{w}}, h \in \mathcal{I}_j^{\mathsf{s}}\}$ to the users. Then, all users can recover their desired computations $\{\boldsymbol{W}\boldsymbol{x}_i\}$ and the scheme preserves privacy against any set of z colluding ENs.*

The proof of Theorem 2 is given in Appendix B. We provide a sense of the proof with the following example.

Example 1. *Consider $e = n = 5$, $p = 3$, and $\pi = (1\ 4\ 2\ 5\ 3)$, the generator of a cyclic permutation group of order* 5*. From* (I.3) *and* (I.5)*, we have*

$$\boldsymbol{I}_{\mathsf{w}} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 1 & 2 & 3 \\ 2 & 3 & 4 & 5 & 1 \end{pmatrix} \text{ and } \boldsymbol{I}_{\mathsf{s}} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 1 \end{pmatrix}.$$

We focus on the matrix of shares $\boldsymbol{S}^{(1)}$. It is assigned to EN e_1 and gets multiplied with the submatrices of $\boldsymbol{W}$ indexed by the elements in the set

$$\mathcal{I}_1^{\mathsf{w}} = \{\pi^0(1), \pi(1), \pi^2(1)\} = \{1, 4, 2\}.$$

Note that the set $\mathcal{I}_1^{\mathsf{w}}$ contains three recursively π-permuted integers of 1 ($\pi^0(1)$, $\pi^1(1)$, and $\pi^2(1)$). Now, consider EN e_5, which is also assigned the matrix of shares $\boldsymbol{S}^{(1)}$. We have

$$\mathcal{I}_5^{\mathsf{w}} = \{\pi^0(5), \pi(5), \pi^2(5)\} = \{5, 3, 1\}.$$

Notice that $\pi^0(5) = \pi^3(1) = 5$ is the fourth (including π^0) recursively π-permuted integer of 1. Hence, the set $\mathcal{I}_1^{\mathsf{w}} \cup \mathcal{I}_5^{\mathsf{w}}$ contains in total six recursively π-permuted integers of 1, which is sufficient to give the set [5], since the group generated by π is transitive. In a similar way, it can be shown that the same property holds for all other matrices of shares. Each matrix of shares is multiplied with all submatrices of $\boldsymbol{W}$, and the sets in (I.2) *are obtained.*

4 Communication and Computation Scheduling, and Private Coding Scheme Optimization

In this section, we describe the scheduling of the proposed scheme. This encompasses the upload of the shares to the ENs, the order of the computations performed at the ENs, the download of a sufficient subset of $\{\boldsymbol{W}_\ell\boldsymbol{S}^{(h)} \mid \ell \in \mathcal{I}_j^{\mathsf{w}}, h \in$

$\mathcal{I}_j^{\mathsf{s}}, j \in [e]\}$, and the decoding of this subset such that each user u_i obtains the desired result $\boldsymbol{y}_i = \boldsymbol{W}\boldsymbol{x}_i$. In the following, we refer to a product $\boldsymbol{W}_\ell \boldsymbol{S}^{(h)}$ as an intermediate result (IR).

4.1 Upload and Computation

Our scheme starts with the users uploading their shares to the ENs. As $\boldsymbol{W}$ stays constant over a long period of time, we assume that it can be stored at the ENs prior to the beginning of the online phase. The users start by sequentially unicasting their shares to the e ENs. Note that, unlike in the nonprivate scheme in [17], the users cannot broadcast their data in the clear—to attain privacy, it needs to be ensured that any z potentially compromised ENs do not gain access to more than $k-1$ distinct shares of the users' private data. Recall that transmission of one element of GF(q) from each user takes γ normalized time units (see Section 2.2). Consequently, it takes γr time units until an EN receives a matrix of shares $\boldsymbol{S}^{(h)}$. The upload schedule is depicted in blue in Fig. I.2. The users first upload their first matrix of shares to EN e_1 and continue with $\mathsf{e}_2, \mathsf{e}_3, \ldots$ sequentially until each EN has received its first matrix of shares. The users then transmit their second matrix of shares to the e ENs, starting with e_1. This continues until each EN has received a matrices of shares; EN e_j receives $\{\boldsymbol{S}^{(h)} \mid h \in \mathcal{I}_j^{\mathsf{s}}\}$. Hence, EN e_j receives its h'-th matrix of shares, $\boldsymbol{S}^{(\phi_j^{\mathsf{s}}(h'))}$, at normalized time

$$\mathsf{L}_j^{\mathsf{up},h'} = \gamma r(e(h'-1)+j)\,,$$

and the total normalized upload latency of the private scheme becomes

$$\mathsf{L}_{\mathsf{P}}^{\mathsf{up}} = \gamma \cdot r \cdot e \cdot a\,.$$

The computation phase at an EN starts as soon as the EN receives the first matrix of shares from the users. Recall from Section 2.1 that the random setup time for EN e_j is λ_j, i.e., EN e_j starts the computation λ_j/τ normalized time units after receiving its first matrix of shares. The setup times are illustrated in red in Fig. I.2. In total, p IRs of the form $\boldsymbol{W}_\ell \boldsymbol{S}^{(h)}$ have to be computed for each assigned matrix of shares $\boldsymbol{S}^{(h)}$ by EN e_j, $j \in [e]$, where $\ell \in \mathcal{I}_j^{\mathsf{w}}$ and $h \in \mathcal{I}_j^{\mathsf{s}}$. This incurs a normalized latency of $p \cdot m/e$, because each $\boldsymbol{W}_\ell$ has m/e rows, and hence the ENs compute $u \cdot m/e$ inner products for each of the p IRs.

It can happen that an EN has not received the next matrix of shares when it finished the computation on the current matrix of shares. In this case, the EN remains idle until the users upload the next matrix of shares. We depict this in yellow in Fig. I.2. For $h' \in [a]$, the normalized time at which EN e_j starts to compute on the h'-th assigned matrix of shares, i.e., on $\boldsymbol{S}^{(\phi_j^{\mathsf{s}}(h'))}$, is

$$\mathsf{L}_j^{\mathsf{start},h'} = \max\left\{\mathsf{L}_j^{\mathsf{start},h'-1} + p\frac{m}{e}\,,\ \mathsf{L}_j^{\mathsf{up},h'}\right\},\ \text{for } h' > 1\,,$$

and

$$\mathsf{L}_j^{\mathsf{start},1} = \frac{\lambda_j}{\tau} + \mathsf{L}_j^{\mathsf{up},1}\,.$$

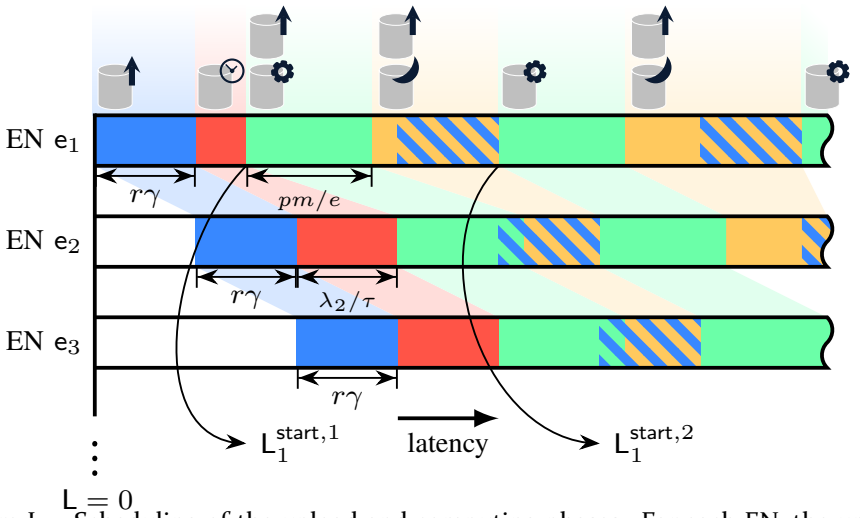


Figure I.2: Scheduling of the upload and computing phases. For each EN, the upload normalized times $r\gamma$ are shown in blue, the random setup times in red, the times pm/e to compute p IRs in green, and possible idle times in yellow.

The computation phase continues at least until the computations in (I.2) are obtained, i.e., until there are at least k distinct IRs of the form $\boldsymbol{W}_\ell \boldsymbol{S}^{(h)}$ for each $\ell \in [e]$. This ensures that user u_i can recover $\boldsymbol{W}\boldsymbol{x}_i$. We remark that it can be beneficial to continue computing products to reduce the communication latency in the download phase, as we discuss next.

4.2 Download

For the download, we exploit zero-forcing precoding to serve multiple users simultaneously and hence reduce the communication latency. An IR $\boldsymbol{W}_\ell \boldsymbol{S}^{(h)}$ that is available at $\rho_{\ell,h}$ ENs incurs a normalized communication latency of $(m/e) \cdot \gamma / \min\{\rho_{\ell,h}, u\}$ (see Section 2.2). Consequently, a high multiplicity of an IR reduces its corresponding communication latency. However, a high multiplicity implies that the same IR has to be computed multiple times at different ENs, thereby increasing the computation latency. There is therefore a tradeoff between communication latency and computation latency, which can be optimized to reduce the overall latency. Assume the optimum is reached after EN e_{j^*} has computed the IR $\boldsymbol{W}_{\phi^{\mathsf{w}}_{j^*}(\ell^*)} \boldsymbol{S}^{(\phi^{\mathsf{s}}_{j^*}(h^*))}$. This gives a normalized computation latency of

$$\mathsf{L}^{\text{comp}} = \mathsf{L}_{j^*}^{\text{start},h^*} + \ell^* \frac{m}{e}.$$

Subsequently, the ENs jointly transmit a subset of the computed IRs $\{\boldsymbol{W}_\ell \boldsymbol{S}^{(h)}\}$ to multiple users simultaneously in descending order of their multiplicity $\rho_{\ell,h}$ until enough results are available to the users such that the sufficient recovery condition in Corollary 1 is met. More precisely, for each $\boldsymbol{W}_\ell$ the ENs send the k IRs with highest multiplicity to the users, thereby ensuring that each user u_i can recover

the desired result $\boldsymbol{W}\boldsymbol{x}_i$. For a fixed ℓ, let

$$\mathcal{H}_\ell^{\max} = \arg \max_{\mathcal{A}\subseteq[n],|\mathcal{A}|=k} \sum_{h\in\mathcal{A}} \rho_{\ell,h}$$

be the set of indices h of the k largest $\rho_{\ell,h}$. Then, the aforementioned download strategy results in a normalized communication latency of

$$\mathsf{L}^{\text{comm}} = \gamma \frac{m}{e} \sum_{\ell=1}^{e} \sum_{h\in\mathcal{H}_\ell^{\max}} \frac{1}{\min\{\rho_{\ell,h}, u\}}.$$

4.3 Decoding Latency

After the users have downloaded a sufficient number of IRs (k IRs for each $\boldsymbol{W}_\ell$), the users need to decode the SSS scheme to obtain their desired results $\{\boldsymbol{y}_i = \boldsymbol{W}\boldsymbol{x}_i\}$. Decoding the SSS scheme means decoding the corresponding RS code. Here, we assume decoding via the Berlekamp-Massey algorithm, which, for an (n, k) RS code, entails $n(n-k)$ multiplications and $n(n-k-1)$ additions [30], plus an additional discrete Fourier transformation that involves $n/2(\lceil\log_2(n)\rceil - 1)$ multiplications and $n\lceil\log_2(n)\rceil$ additions [31, Eq. (8)]. We assume that it takes the same time to perform one addition and one multiplication, i.e., both operations take the same amount of clock cycles. This assumption is reasonable, as both operations can be performed using either a look-up table or, in case q is a prime, integer arithmetic in the arithmetic and logic units of the user devices' processors. We make this assumption because it significantly simplifies the analysis. Recall that a user requires δ normalized time units to compute an inner product in $\mathrm{GF}(q)^r$, which comprises r multiplications and $r-1$ additions in $\mathrm{GF}(q)$. The latency of performing an addition or a multiplication is hence $\delta/(2r-1)$. With this, the decoding latency for each user can be written in closed-form as

$$\mathsf{L}^{\text{dec}} = \frac{\delta}{2r-1} mn\left(2(n-k) + \frac{3}{2}\lceil\log_2(n)\rceil - \frac{3}{2}\right),$$

since the users have to perform

$$\frac{m}{e} \cdot e \cdot n\left(2(n-k) + \frac{3}{2}\lceil\log_2(n)\rceil - \frac{3}{2}\right)$$

operations in $\mathrm{GF}(q)$ (one RS decoding per row) for each of the e matrices $\{\boldsymbol{W}_\ell\}$ while needing $\delta/(2r-1)$ normalized times units per operation.

The overall normalized latency becomes

$$\begin{aligned}\mathsf{L} &= \mathsf{L}^{\text{comp}} + \mathsf{L}^{\text{comm}} + \mathsf{L}^{\text{dec}} \\ &= \mathsf{L}_{j^*}^{\text{start},h^*} + \ell^* \frac{m}{e} + \gamma \frac{m}{e} \sum_{\ell=1}^{e} \sum_{h\in\mathcal{H}_\ell^{\max}} \frac{1}{\min\{\rho_{\ell,h}, u\}} \\ &\quad + \frac{\delta}{2r-1} mn\left(2(n-k) + \frac{3}{2}\lceil\log_2(n)\rceil - \frac{3}{2}\right). \end{aligned} \tag{I.8}$$

4.4 Private Coding Scheme Optimization

The system design includes the SSS code, which we denote by $\mathscr{C}_{\mathsf{SSS}}$, the assignment matrices $\boldsymbol{I}_{\mathsf{w}}$ and $\boldsymbol{I}_{\mathsf{s}}$, the number of ENs over which the users offload the linear inference operation, $e \leq e_{\mathsf{max}}$, and the privacy level z. To construct matrices $\boldsymbol{I}_{\mathsf{w}}$ and $\boldsymbol{I}_{\mathsf{s}}$, we require a permutation group generator π and the parameter p. Further, to determine a good stopping point for the computation phase, we introduce the parameter t, defined as the number of (not necessarily distinct) IRs for each $\boldsymbol{W}_\ell$ computed across all ENs to wait for before the download phase starts. Note that t should be such that the ENs have collected enough distinct IRs so that the users can decode to recover $\{\boldsymbol{y}_i = \boldsymbol{W}_i \boldsymbol{x}_i\}$. However, it might be useful to collect more IRs than the minimum necessary to reduce the communication latency. As soon as there are t (not necessarily distinct) IRs computed across all ENs for each submatrix of $\boldsymbol{W}$, the ENs stop the computation and begin the download phase.

We refer to the tuple $(\mathscr{C}_{\mathsf{SSS}}, e, \pi, p, t, z)$ as the *private coding scheme*. The goal is then to optimize the private coding scheme, i.e., the above-mentioned tuple, in order to minimize L in (I.8) for a given privacy level z.

Note that $e \leq e_{\mathsf{max}}$ (it may be beneficial to contact less ENs than the ones available). Furthermore, even for the lowest level of privacy, $z = 1$, the users need to contact at least 2 ENs, i.e., $2 \leq e \leq e_{\mathsf{max}}$. Additionally, $1 \leq p \leq \lfloor \mu e \rfloor$; each EN needs to be assigned at least one partition of $\boldsymbol{W}$ and it may be beneficial that the ENs do not utilize their whole storage capacity, because storing fewer than $\lfloor \mu e \rfloor$ submatrices of $\boldsymbol{W}$ leads to the ENs performing computations on the later shares sooner. Lastly, there are some constraints on the parameters n and k of the SSS code $\mathscr{C}_{\mathsf{SSS}}$. From the SSS code, it follows that $n \geq k$, whereas from the combinatorial design $n \leq e$. The value of k depends on the desired privacy level z and the number of matrices of shares each EN has access to, a (which follows from e, p, and n, see (I.7)). In the worst case, z ENs have access to az distinct shares. Therefore, we need $k \geq az + 1$ to ensure privacy. Consequently, we get $az + 1 \leq n \leq e$. Note that there is no reason to select $k > az + 1$ as this leads to reduced straggler mitigation and increased computational load.

5 Variants

In this section, we introduce two variants to the private scheme proposed in Sections 3 and 4. First, we notice that we can reduce the overall latency by starting the download as soon as the upload phase is completed, i.e., the download phase and the computation phase can be performed simultaneously. We propose to use a priority queue that determines the order in which computed IRs should be downloaded. Secondly, we introduce an additional layer of coding to the scheme by encoding the network-side matrix $\boldsymbol{W}$ prior to storing it over the ENs. We also relax some of the constraints on the system parameters.

5.1 Priority Queue

Instead of waiting for the computation phase to finish, IRs can be downloaded as soon as they are available and the channel is idle, i.e., when the upload phase is completed and no other IR is being downloaded. To determine which IR to send, we equip the ENs with a shared priority queue in which the pairs of indices that identify the IRs, (ℓ, h) (where ℓ identifies the partition of $\boldsymbol{W}$, $\boldsymbol{W}_\ell$, and h the matrix of shares, $\boldsymbol{S}^{(h)}$), are queued. A priority queue is a data structure in which each element has an associated priority. Elements with high priority will leave the queue before elements with low priority. Particularly, we consider a priority queue in which the priority is given by the multiplicity of an IR. After an EN has finished the computation of an IR, it either adds the corresponding pair of indices (ℓ, h) to the queue or increments its multiplicity (priority) if it already exists in the queue. Anytime the channel is available and there are index pairs left in the queue, the ENs cooperatively send the corresponding IR with the highest priority (i.e., highest multiplicity) to the users. This ensures that at any time the ENs send the IR with the lowest associated communication cost to the users. In contrast to the scheme in Sections 3 and 4, there is no optimization needed to determine t, as the download starts as soon as the upload phase finishes. Hence, the optimization is over $(\mathscr{C}_{\mathsf{SSS}}, e, \pi, p, z)$ for a given value of z. Further, the ENs have to keep track of the queue and its complete history. This way, already downloaded IRs do not need to be computed again.

5.2 Additional Coding on the Network-Side Matrix $\boldsymbol{W}$

The straggler resiliency of the scheme proposed in the previous sections can be increased by introducing an additional layer of coding on the matrix $\boldsymbol{W}$. In particular, we partition $\boldsymbol{W}$ row-wise into k' submatrices and encode it using an (n', k') RS code, denoted by $\mathscr{C}_{\mathsf{w}}$. We denote by $\boldsymbol{C} = \left(\boldsymbol{C}_1^\top, \boldsymbol{C}_2^\top, \ldots, \boldsymbol{C}_{n'}^\top\right)^\top$ the resulting coded matrix, comprising n' submatrices. The n' coded submatrices of $\boldsymbol{C}$ are then assigned to the ENs. Compared to the uncoded case, we relax the condition that the number of submatrices equals the number of ENs e. For $n' = e$, the same assignment of submatrices to ENs as the one used in Section 3.2 for the uncoded matrix $\boldsymbol{W}$ can be used. For $n' \neq e$, however, we need to modify the assignment. For $n' \geq e$, we simply take a cyclic permutation group π_{c} of order n' to fill the index matrix $\boldsymbol{I}_{\mathsf{c}}$ that determines the assignment of submatrices $\{\boldsymbol{C}_\ell\}$ to ENs (i.e., $\boldsymbol{I}_{\mathsf{c}}$ is the counterpart of $\boldsymbol{I}_{\mathsf{w}}$ for the uncoded case and π_{c} is the counterpart of π, see Section 3.2). Using the same approach for $n' < e$ works, but it leads to a nonuniform distribution of indices in $\boldsymbol{I}_{\mathsf{c}}$. This would lead to higher multiplicity for some IRs, which is suboptimal in terms of download latency. Increasing the multiplicity of IRs has diminishing returns; increasing the multiplicity from 1 to 2 reduces the communication latency by 50%, whereas increasing the multiplicity from 2 to 3 yields a decrease of only 33.3%. This means that the highest gains are obtained by increasing the multiplicity simultaneously across IRs, i.e., we are interested in obtaining a distribution of indices in $\boldsymbol{I}_{\mathsf{c}}$ as close as possible to a uniform distribution. To accomplish that, we propose the following index assignment for $n' < e$.

We start by cyclically filling $\boldsymbol{I}_{\mathrm{c}}$ with indices in $[n']$,

$$\boldsymbol{I}_{\mathrm{c}} = \begin{pmatrix} 1 & 2 & \cdots & n' & ? & ? & \cdots & ? \\ ? & 1 & 2 & \cdots & n' & ? & \cdots & ? \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ ? & \cdots & ? & 1 & 2 & \cdots & n' & ? \end{pmatrix}. \tag{I.9}$$

The left-out entries marked with ? are filled such that the distribution of indices in $\boldsymbol{I}_{\mathrm{c}}$ is as close to uniform as possible while not repeating indices in the same column of $\boldsymbol{I}_{\mathrm{c}}$, as this assignment does not favor a specific submatrix of $\boldsymbol{C}$ and prevents the same submatrix being assigned twice to one EN. Note that (I.9) is only one example of how $\boldsymbol{I}_{\mathrm{c}}$ can look like. Depending on e, p, and n' there might be wrap-arounds of indices.

We can also relax the condition that the number of secret shares has to be less than or equal to the number of ENs, i.e., we allow $n > e$, and simply consider a permutation group π_{s} of order $\max(n, e)$ and construct $\boldsymbol{I}_{\mathrm{s}}$ as in (I.5) (with $\pi = \pi_{\mathrm{s}}$).

Remark 1. *By allowing $n' \neq e$ and $n > e$, it becomes difficult to prove a similar result as in Corollary 1 for the uncoded case on a sufficient condition on the cardinality of $\mathcal{I}$ and $\mathcal{J}$ such that the linear inference can be completed from the IRs $\{\boldsymbol{C}_\ell \boldsymbol{S}^{(h)} \mid h \in \mathcal{I}, \ell \in \mathcal{J}\}$. However, our numerical results reveal that encoding $\boldsymbol{W}$ and relaxing the constraints $n' = e$ and $n \leq e$ allows to reduce the overall latency compared to the scheme in Sections 3 and 4.*

For each user u_i—with its private data $\boldsymbol{x}_i$ and set of random vectors $\{\boldsymbol{r}_i^{(1)}, \ldots, \boldsymbol{r}_i^{(k-1)}\}$—the combination of the (n, k) RS code on $\{\boldsymbol{x}_i, \boldsymbol{r}_i^{(1)}, \ldots, \boldsymbol{r}_i^{(k-1)}\}$ used in the SSS scheme and the (n', k') RS code on $\{\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{k'}\}$ can be seen as an (nn', kk') product code (with nonsystematic component codes) over $\{\boldsymbol{W}_\ell \boldsymbol{x}_i | \ell \in [k']\}$ (i.e., the desired inference $\boldsymbol{W}\boldsymbol{x}_i$) and $\{\boldsymbol{W}_\ell \boldsymbol{r}_i^{(\kappa)} | \ell \in [k'], \kappa \in [k-1]\}$. To show this, we arrange the elements of $\{\boldsymbol{C}_\ell \boldsymbol{s}_i^{(h)}\}$ in the $n' \times n$ two-dimensional array

$$\begin{bmatrix} \boldsymbol{C}_1 \boldsymbol{s}_i^{(1)} & \boldsymbol{C}_1 \boldsymbol{s}_i^{(2)} & \cdots & \boldsymbol{C}_1 \boldsymbol{s}_i^{(n)} \\ \boldsymbol{C}_2 \boldsymbol{s}_i^{(1)} & \boldsymbol{C}_2 \boldsymbol{s}_i^{(2)} & \cdots & \boldsymbol{C}_2 \boldsymbol{s}_i^{(n)} \\ \vdots & \cdots & \ddots & \vdots \\ \boldsymbol{C}_{n'} \boldsymbol{s}_i^{(1)} & \boldsymbol{C}_{n'} \boldsymbol{s}_i^{(2)} & \cdots & \boldsymbol{C}_{n'} \boldsymbol{s}_i^{(n)} \end{bmatrix}.$$

It is easy to see that each row of the array is a codeword of an (n, k) code and each column is a codeword of an (n', k') code. More precisely, $(\boldsymbol{s}_i^{(1)}, \ldots, \boldsymbol{s}_i^{(n)})$ is the codeword corresponding to the encoding of $(\boldsymbol{x}_i, \boldsymbol{r}_i^{(1)}, \ldots, \boldsymbol{r}_i^{(k-1)})$ via the SSS (n, k) RS code. Since the RS code is linear, $(\boldsymbol{C}_\ell \boldsymbol{s}_i^{(1)}, \ldots, \boldsymbol{C}_\ell \boldsymbol{s}_i^{(n)})$ is also a codeword of an (n, k) RS code, which would result from encoding $(\boldsymbol{C}_\ell \boldsymbol{x}_i, \boldsymbol{C}_\ell \boldsymbol{r}_i^{(1)}, \ldots, \boldsymbol{C}_\ell \boldsymbol{r}_i^{(k-1)})$. Likewise, $(\boldsymbol{C}_1, \ldots, \boldsymbol{C}_{n'})$ is the codeword corresponding to the encoding of $(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{k'})$ via the (n', k') RS code on $\boldsymbol{W}$, and $(\boldsymbol{C}_1 \boldsymbol{s}_i^{(h)}, \ldots, \boldsymbol{C}_{n'} \boldsymbol{s}_i^{(h)})$ is a codeword of an (n', k') RS code corresponding to the encoding of $(\boldsymbol{W}_1 \boldsymbol{s}_i^{(h)}, \ldots, \boldsymbol{W}_{k'} \boldsymbol{s}_i^{(h)})$.

The product code structure allows the users to iteratively decode the received results, which provides more flexibility regarding the decodable patterns; there

are sets of IRs that allow to complete the linear inference operation by iterating between row and column decoders, while either component code would fail to decode on its own. To illustrate the iterative decoding procedure, we provide the following example.

Example 2. *Consider the SSS* $(n,k) = (4,3)$ *RS code and an* $(n',k') = (3,2)$ *RS code on* $\boldsymbol{W}$. *Encode* $\boldsymbol{W}$ *into a matrix* $\boldsymbol{C}$ *and arrange all* $\{\boldsymbol{C}_\ell \boldsymbol{s}_i^{(h)} | \ell \in [3], h \in [4]\}$ *in an array of dimensions* 3×4 *to show the product code structure,*

$$\begin{bmatrix} \boldsymbol{C}_1\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_1\boldsymbol{s}_i^{(2)} & \boldsymbol{C}_1\boldsymbol{s}_i^{(3)} & \boldsymbol{C}_1\boldsymbol{s}_i^{(4)} \\ \boldsymbol{C}_2\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(2)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(3)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(4)} \\ \boldsymbol{C}_3\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_3\boldsymbol{s}_i^{(2)} & \boldsymbol{C}_3\boldsymbol{s}_i^{(3)} & \boldsymbol{C}_3\boldsymbol{s}_i^{(4)} \end{bmatrix}.$$

Each row of the array is a codeword of a $(4,3)$ *RS code and each column is a codeword of a* $(3,2)$ *RS code.*

Assume that the users have the following IRs available,

$$\begin{bmatrix} \boldsymbol{C}_1\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_1\boldsymbol{s}_i^{(2)} & & \\ & \boldsymbol{C}_2\boldsymbol{s}_i^{(2)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(3)} & \\ \boldsymbol{C}_3\boldsymbol{s}_i^{(1)} & & & \boldsymbol{C}_3\boldsymbol{s}_i^{(4)} \end{bmatrix}.$$

As we can see, there are no $k = 3$ *IRs available for any* $\boldsymbol{C}_\ell$. *Therefore, the users would not be able to decode any SSS scheme. However, we have* $k' = 2$ *IRs available in the first and second column. The users can then decode the column RS code for columns one and two to obtain* $\boldsymbol{C}_2\boldsymbol{s}_i^{(1)}$ *and* $\boldsymbol{C}_3\boldsymbol{s}_i^{(2)}$,

$$\begin{bmatrix} \boldsymbol{C}_1\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_1\boldsymbol{s}_i^{(2)} & & \\ \boldsymbol{C}_2\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(2)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(3)} & \\ \boldsymbol{C}_3\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_3\boldsymbol{s}_i^{(2)} & & \boldsymbol{C}_3\boldsymbol{s}_i^{(4)} \end{bmatrix}.$$

Now, there are $k = 3$ *IRs available in the second and third row, hence the users can decode the corresponding row codes to obtain* $\boldsymbol{C}_2\boldsymbol{s}_i^{(4)}$ *and* $\boldsymbol{C}_3\boldsymbol{s}_i^{(3)}$,

$$\begin{bmatrix} \boldsymbol{C}_1\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_1\boldsymbol{s}_i^{(2)} & & \\ \boldsymbol{C}_2\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(2)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(3)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(4)} \\ \boldsymbol{C}_3\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_3\boldsymbol{s}_i^{(2)} & \boldsymbol{C}_3\boldsymbol{s}_i^{(3)} & \boldsymbol{C}_3\boldsymbol{s}_i^{(4)} \end{bmatrix}.$$

Lastly, the users can switch to column decoding again as now there are $k' = 2$ *IRs available in the third and fourth column, and the whole code array can be recovered,*

$$\begin{bmatrix} \boldsymbol{C}_1\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_1\boldsymbol{s}_i^{(2)} & \boldsymbol{C}_1\boldsymbol{s}_i^{(3)} & \boldsymbol{C}_1\boldsymbol{s}_i^{(4)} \\ \boldsymbol{C}_2\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(2)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(3)} & \boldsymbol{C}_2\boldsymbol{s}_i^{(4)} \\ \boldsymbol{C}_3\boldsymbol{s}_i^{(1)} & \boldsymbol{C}_3\boldsymbol{s}_i^{(2)} & \boldsymbol{C}_3\boldsymbol{s}_i^{(3)} & \boldsymbol{C}_3\boldsymbol{s}_i^{(4)} \end{bmatrix}.$$

At last, the users are able to recover all IRs and thereby the computations $\{\boldsymbol{W}\boldsymbol{x}_i\}$. *For this particular example, this would not have been possible without the redundancy on the submatrices of* $\boldsymbol{W}$.

The private coding scheme with coding over $\boldsymbol{W}$ and priority queue is defined by the tuple $(\mathscr{C}_{\mathsf{SSS}}, \mathscr{C}_{\mathsf{w}}, e, \pi_{\mathsf{c}}, \pi_{\mathsf{s}}, p, z)$, which should be properly optimized for a given privacy level z.

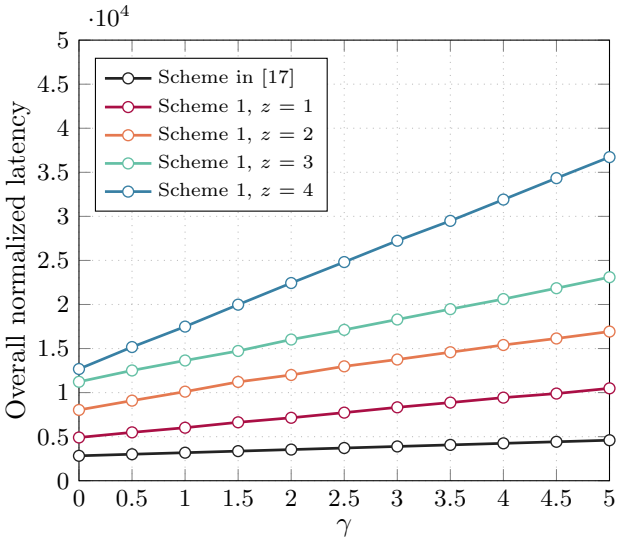


Figure I.3: Expected overall normalized latency as a function of γ for different privacy levels z of the proposed scheme (Scheme 1) compared to the nonprivate scheme in [17]. The parameters are $\mu = 2/3$, $\tau = 0.0005$, $\eta = 0.5$, $e_{\max} = 9$, $m = 600$, $r = 50$, and $\delta = 3$.

6 Numerical Results

In this section, we compare the performance of the proposed private scheme in Sections 3 and 4, and its variants in Section 5, with that of the nonprivate scheme in [17]. For convenience, we will refer to the scheme in Sections 3 and 4 as Scheme 1, and to the two variants in Section 5 as Scheme 2 (Scheme 1 augmented with a priority queue) and Scheme 3 (Scheme 2 augmented with coding over $\boldsymbol{W}$).

For all numerical results, the maximum number of ENs is $e_{\max} = 9$, the storage capacity is $\mu = 2/3$, $\boldsymbol{W}$ has dimensions 600×50, the computation time is $\tau = 0.0005$, and the straggling parameter is $\eta = 0.5$. Lastly, we assume that the users are $\delta = 3$ times slower than the ENs. Note that due to the normalization by τ, the number of users is inconsequential on the normalized overall latency L as long as $u \geq \max_{\ell,h} \rho_{\ell,h}$, e.g., if $u \geq e$. In the simulations we consider $u \geq e$, which is usually the case in practice.

For the optimization of the coding schemes, we fix the generator of the cyclic permutation group to $\pi = (1 \ e \ e-1 \ \cdots \ 2)$ for Schemes 1 and 2 whereas for Scheme 3 we vary n' and assign the submatrices $\boldsymbol{C}_\ell$ as described in Section 5.2. For Scheme 3, we use $\pi_{\mathsf{s}} = (1 \ \max(n,e) \ \max(n,e)-1 \ \cdots \ 2)$ and if $n' \geq e$, we use $\pi_{\mathsf{c}} = (1 \ n' \ n'-1 \ \cdots \ 2)$. We then optimize the other parameters for a given privacy level z. Particularly, we perform an exhaustive search over all feasible parameter values. For each set of parameters, unless otherwise stated, we generated 10^4 instances of the random setup times $\{\lambda_j\}$ and simulated the scheme. We then select the parameters that yield the best expected overall latency over the 10^4 runs.

In Fig. I.3, we plot the expected overall latency $\mathbb{E}[\mathsf{L}]$ (given by (I.8)) as a function of γ for Scheme 1 with different values of z and compare its performance to that of

the nonprivate scheme in [17]. We remark that in [17] both the upload latency and the decoding latency are neglected, while we consider them here. For the scheme in [17], we assume as in [17] that the users can broadcast their local data to all ENs simultaneously. However, in general, broadcasting a message to e receivers is more expensive than transmitting a single unicast message to one receiver. As in [4], we assume that broadcasting to e receivers is a factor $\log(e)$ more expensive in terms of latency than a single unicast. Recall that the normalized latency of unicasting u vectors from $\mathrm{GF}(q)^r$ is γr. Hence, for the nonprivate scheme in [17], the normalized latency of every user broadcasting its local data to all e ENs is $\mathsf{L}_{\mathsf{NP}}^{\mathsf{up}} = \gamma \cdot r \cdot \log(e)$.

To yield privacy, the proposed scheme involves more communication and computation at the ENs than the nonprivate scheme, as there are multiple shares to be transmitted and computed on instead of a single vector $\boldsymbol{x}_i$ per user. As a result, the proposed scheme has a higher latency. As expected, the expected overall latency increases with the privacy level z. For $\gamma = 0$, the latency of the private scheme increases by a factor 1.7, 2.8, 4.0, and 4.5 for $z = 1$, 2, 3, and 4, respectively, compared to the nonprivate scheme, whereas for $\gamma = 5$ the factors are 2.3, 3.7, 5.0, and 8.0, respectively. The relative increase in latency increases with γ (i.e., increases with the relative communication costs) due to the aforementioned higher communication load of the proposed scheme. We also notice that the proposed scheme does not always utilize all available ENs. For example, for $z = 1$ and $\gamma = 2.5$, Scheme 1 has the lowest expected overall latency when contacting only $e = 6$ ENs. The parameter e influences not only the upload cost, but also the number of submatrices of $\boldsymbol{W}$, which in turn influences the number of submatrices stored at each EN, p, which effects the multiplicity of IRs. This complex interplay of dependencies on e makes it difficult to predict the optimal value of e. For example, for $z = 1$, the optimal e increases with γ (we have $e = 8$ for $\gamma \geq 4$) whereas for $z = 2$, e decreases with γ (from $e = 9$ for $\gamma \leq 1.5$ to $e = 8$ for $\gamma \geq 2$).

In Fig. I.4, we compare the performance of Scheme 1 with that of Scheme 2 and the nonprivate scheme in [17]. The use of a priority queue reduces the expected overall latency, especially for high values of γ, i.e., when communication is comparatively expensive. As a result, for $z = 1$, Scheme 2 performs similar to the nonprivate scheme, while providing privacy against one honest-but-curious server.

In Fig. I.5, we plot the expected overall latency $\mathbb{E}[\mathsf{L}]$ versus γ for Scheme 2, Scheme 3, and the scheme in [17]. The higher flexibility offered by adding redundancy on $\boldsymbol{W}$ allows to further reduce the expected overall latency with respect to Scheme 2 for low values of γ, for which the computation times dominate and straggler mitigation is important. Interestingly, this improvement allows the private scheme to outperform the nonprivate scheme for $z = 1$. This is explained by the high decoding cost of the scheme in [17] compared to the proposed scheme. Indeed, the RS code used in the SSS scheme has very small length and dimension, whereas the MDS code used in [17] has much higher length and dimension. For example, for $\gamma = 1$ with Scheme 3 and $z = 1$ we have $(n', k') = (4, 3)$ and for the nonprivate scheme the code length and dimension are in the order of m ($m = 600$ in this scenario). Therefore, the nonprivate scheme suffers from higher decoding latency, which significantly penalizes the expected overall latency. For high values

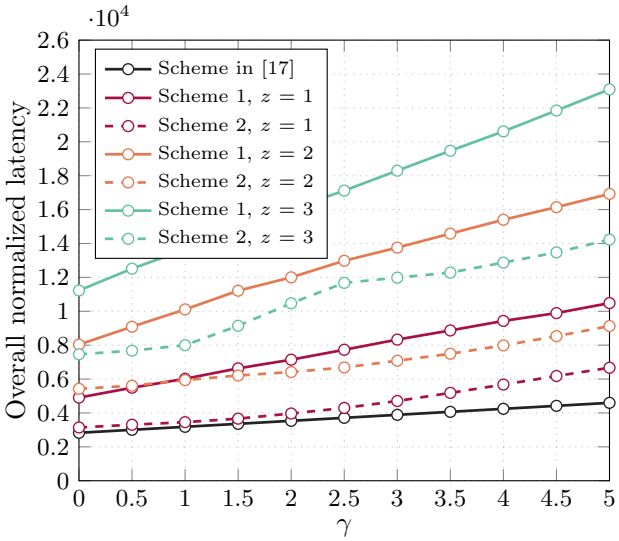


Figure I.4: Expected overall normalized latency as a function of γ for different privacy levels z of the proposed scheme (Scheme 1) compared to the priority queue variant (Scheme 2) and the nonprivate scheme in [17]. The parameters are $\mu = 2/3$, $\tau = 0.0005$, $\eta = 0.5$, $e_{\max} = 9$, $m = 600$, $r = 50$, and $\delta = 3$.

of γ, i.e., when the communication latency becomes more critical, it is beneficial to use as much replication as possible to increase the multiplicities of the IRs to reduce the communication latency in the download. This means that small n and n' are beneficial to reduce the number of distinct IRs. As a consequence, coding on $\boldsymbol{W}$ brings almost no improvement for high γ, as we have $n = k$ and $n' = k'$ (i.e., no RS coding) for as low k and k' as possible.

For some applications, the expected overall latency may not be the most relevant performance metric. In Fig. I.6, we consider edge computing under a deadline, where we are interested in completing the linear inference within some overall latency. Particularly, we plot the probability that the linear inference is not completed within a deadline L, for $z = 1$ and $\gamma = 1$ and 4.5. To this end, for a given probability, we optimize over (a subset of) the parameters $(\mathscr{C}_{\mathsf{SSS}}, \mathscr{C}_{\mathsf{w}}, e, \pi, \pi_{\mathsf{c}}, \pi_{\mathsf{s}}, p, t, z)$ for $\pi = (1\ e\ e-1\ \cdots\ 2)$, $\pi_{\mathsf{c}} = (1\ n'\ n'-1\ \cdots\ 2)$, $\pi_{\mathsf{s}} = (1\ \max(n,e)\ \max(n,e)-1\ \cdots\ 2)$, and $z = 1$ to minimize L. The number of samples of $\{\lambda_j\}$ is increased to 10^6 to get reliable results for probabilities down to 10^{-4}.

For $\gamma = 4.5$ and a deadline $\mathsf{L} = 10^4$, the probability of exceeding the deadline is $4.0 \cdot 10^{-1}$ for Scheme 1, while it decreases to $3.9 \cdot 10^{-3}$ for Scheme 2, i.e., two orders of magnitude lower. Introducing coding over $\boldsymbol{W}$ does not bring further gains. For $\gamma = 1$ and a deadline $\mathsf{L} = 10^4$, the probability of exceeding the deadline is $7.2 \cdot 10^{-2}$ for Scheme 1, while it decreases to $4.5 \cdot 10^{-4}$ for Scheme 2. Again, we see an improvement of about two orders of magnitude. Furthermore, for this low value of γ, introducing coding over $\boldsymbol{W}$ reduces the probability of not meeting the deadline further to $9.7 \cdot 10^{-5}$.

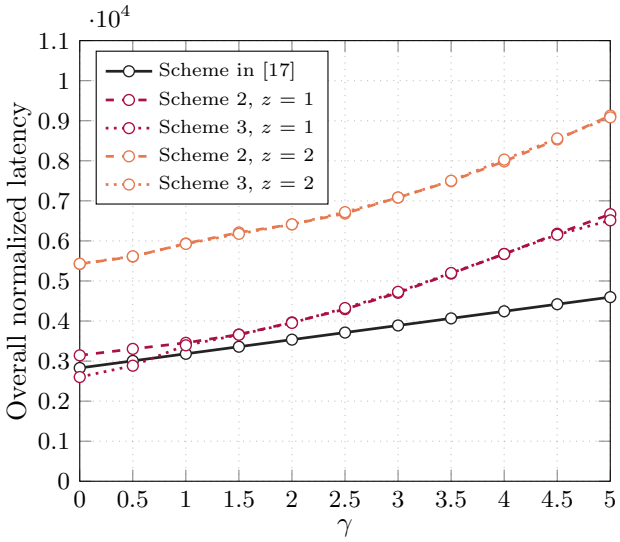


Figure I.5: Expected overall normalized latency as a function of γ for different privacy levels z of the priority queue variant (Scheme 2), the priority queue with coding on $\boldsymbol{W}$ variant (Scheme 3), and the nonprivate scheme in [17]. The parameters are $\mu = 2/3$, $\tau = 0.0005$, $\eta = 0.5$, $e_{\max} = 9$, $m = 600$, $r = 50$, and $\delta = 3$.

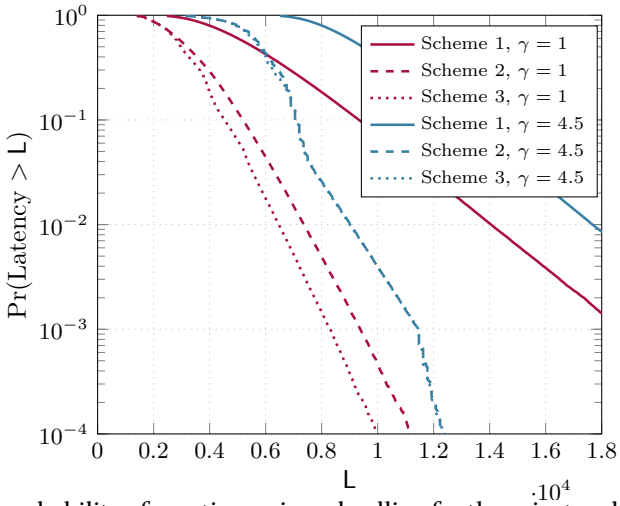


Figure I.6: The probability of meeting a given deadline for the private scheme (Scheme 1) and its variants (Schemes 2 and 3) with $z = 1$ for different values of γ.

7 Conclusion

We introduced three coded edge computing schemes for linear inference at the network edge that provide privacy against up to z colluding edge servers while minimizing the overall latency encompassing upload, computation, download, and decoding latency. The proposed schemes combine secret sharing to provide privacy and straggler resiliency, possibly coding over the network model matrix for further straggler mitigation, and replication of subtasks across edge servers to create cooperation opportunities between edge servers to reduce the download communication latency. Numerical results show that, for a considered scenario with 9 edge servers, the proposed scheme yields a 8% latency reduction compared to the nonprivate scheme by Zhang and Simeone while providing privacy against one honest-but-curious edge server. The privacy level can be enhanced at the expense of a higher latency.

Appendix

A Proof of Theorem 1

Let $\mathscr{C}_{\mathsf{SSS}}$ be the (n, k) RS code used in the SSS scheme. For each $h \in [n]$, the entries of the rows of $\boldsymbol{S}^{(h)}$ are code symbols in position h of codewords from $\mathscr{C}_{\mathsf{SSS}}$ pertaining to different users. More precisely, for each user u_i, each row of the matrix $\left(\boldsymbol{s}_i^{(1)}, \boldsymbol{s}_i^{(2)}, \dots, \boldsymbol{s}_i^{(n)}\right)$ of all n shares of u_i is a codeword from $\mathscr{C}_{\mathsf{SSS}}$. Since $\mathscr{C}_{\mathsf{SSS}}$ is a linear code, each of the m rows of the matrix

$$\boldsymbol{W}\left(\boldsymbol{s}_i^{(1)}, \boldsymbol{s}_i^{(2)}, \dots, \boldsymbol{s}_i^{(n)}\right)$$

is a codeword of $\mathscr{C}_{\mathsf{SSS}}$. Furthermore, the messages obtained by decoding these codewords are the rows of

$$\left(\boldsymbol{W}\boldsymbol{x}_i, \boldsymbol{W}\boldsymbol{r}_i^{(1)}, \dots, \boldsymbol{W}\boldsymbol{r}_i^{(k-1)}\right).$$

Then, decoding the vectors in the set $\{\boldsymbol{W}\boldsymbol{s}_i^{(h)} \mid h \in \mathcal{I}\}$ gives $\boldsymbol{W}\boldsymbol{x}_i$, and it follows that $\{\boldsymbol{W}\boldsymbol{S}^{(h)} \mid h \in \mathcal{I}\}$ gives $\{\boldsymbol{W}\boldsymbol{x}_i\}$.

From the properties of the SSS scheme, it follows that the mutual information between $\{\boldsymbol{S}^{(h)} \mid h \in \mathcal{J}\}$ and $\{\boldsymbol{x}_i\}$ is zero. Subsequently, from the data processing inequality, it follows that $\{\boldsymbol{W}\boldsymbol{S}^{(h)} \mid h \in \mathcal{J}\}$ reveals no information about $\{\boldsymbol{x}_i\}$.

B Proof of Theorem 2

The proof makes heavy use of combinatorics. For readers unfamiliar with this field, especially the nomenclature of blocks and points, we recommend [32]. We define a map $(x)_e$ that maps an integer x onto the set $[e]$ by successively adding or subtracting e to x until the result lies in $[e]$. For example, for $e = 5$, we have $(3)_5 = 3$, $(-2)_5 = 3$, and $(7)_5 = 2$. In contrast to taking a modulo e, we have

$(e)_e = e$, whereas $e \mod e = 0$. The rationale for introducing this map instead of the conventional modulo arithmetic is that the indices of matrix rows and columns run from 1, and not from 0.

We start by proving the recovery ability. $\boldsymbol{I}_{\mathsf{w}}$ is a combinatorial design $\mathfrak{D}$ with e blocks—the e sets with entries from the e columns of $\boldsymbol{I}_{\mathsf{w}}$—and e points—each point is the index ℓ pertaining to the submatrix $\boldsymbol{W}_\ell$. In particular, block j of $\mathfrak{D}$ is $\mathcal{B}_j^{(\mathfrak{D})} = \mathcal{I}_j^{\mathsf{w}}$. Furthermore, each row i of $\boldsymbol{I}_{\mathsf{s}}$ combined with $\boldsymbol{I}_{\mathsf{w}}$ represents a combinatorial design $\mathfrak{D}_i$, where its blocks are a permutation $\pi^{-(i-1)(e-p)}$ of the blocks in $\mathfrak{D}$. More precisely, we have block j of $\mathfrak{D}_i$ as

$$\mathcal{B}_j^{(\mathfrak{D}_i)} = \mathcal{B}_{\pi^{-(i-1)(e-p)}(j)}^{(\mathfrak{D})}.$$

Consider $\boldsymbol{\Delta}^{(\mathfrak{D}_i)}$ to be an incidence matrix, of dimensions $e \times e$, where the incidence relation is between the set of points, $[e]$, and the set of blocks, $\{\mathcal{B}_j^{(\mathfrak{D}_i)} \mid j \in [e]\}$. Then, to prove the recovery ability, we need to show that for

$$\boldsymbol{\Delta} = \sum_{i=1}^{\beta+1} \boldsymbol{\Delta}^{(\mathfrak{D}_i)},$$

we have

$$\delta_{ij} \geq 1, \quad \forall i \in [e],\ j \in [e], \tag{I.10}$$

where δ_{ij} is the element in the i-th row and j-th column of $\boldsymbol{\Delta}$.

We will now show that (I.10) holds. In the construction of $\boldsymbol{I}_{\mathsf{w}}$ and $\boldsymbol{I}_{\mathsf{s}}$ in (I.3) and (I.5), respectively, we consider a cyclic permutation group of order e with elements

$$\pi^0, \pi, \pi^2, \ldots, \pi^{e-1},$$

where π is the generator and π^0 is the identity element of the group. The set

$$\{\pi^0(j), \pi(j), \pi^2(j), \ldots, \pi^{e-1}(j)\} = [e],$$

since π is the generator of the group, and the group is transitive. Let α be the number of cyclic shifts between two consecutive rows of $\boldsymbol{I}_{\mathsf{w}}$. Then,

$$\pi^i(j) = (j + i(e-\alpha))_e = (j - i\alpha)_e,$$

where $i \in [e]$. Note that the blocks of $\mathfrak{D}$ are

$$\mathcal{B}_j^{(\mathfrak{D})} = \{\pi^0(j), \pi(j), \ldots, \pi^{p-1}(j)\}.$$

We see that block j consists of p consecutive permutations of j. Furthermore, for $d \in [\beta]$,

$$\begin{aligned}\pi^{-d(e-p)}(j) &= (j - d(e-\alpha)(e-p))_e \\ &= (j - dp\alpha)_e.\end{aligned}$$

In other words, $\pi^{-d(e-p)} = \pi^{dp}$. Thus, for some $j \in [e]$, we have

$$\mathcal{B}^{(\mathfrak{D})}_{\pi^{-d(e-p)}(j)} = \{\pi^{dp}(j), \pi^{dp+1}(j), \dots, \pi^{(d+1)p-1}(j)\},$$

form which it follows that

$$\mathcal{B}^{(\mathfrak{D})}_{j} \cup \left(\bigcup_{d=1}^{\beta} \mathcal{B}^{(\mathfrak{D})}_{\pi^{-d(e-p)}(j)} \right) = [e].$$

Notice that $\mathcal{B}^{(\mathfrak{D})}_{\pi^{-d(e-p)}(j)}$ is the support of $\boldsymbol{\delta}^{(\mathfrak{D}_{d+1})}_{j}$, the j-th column of $\boldsymbol{\Delta}^{(\mathfrak{D}_{d+1})}$. Thus, (I.10) holds.

The privacy of the scheme follows straightforwardly. Any z colluding ENs have access to at most az distinct matrices of shares. Since we have $k \geq az+1$, it follows from Theorem 1 that the user data privacy is guaranteed.

References

[1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Mar. 2017.

[2] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing–a key technology towards 5G," *ETSI white paper*, no. 11, pp. 1–16, Sep. 2015.

[3] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Washington, DC, USA, Dec. 2016, pp. 1–6.

[4] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

[5] A. Severinson, A. Graell i Amat, and E. Rosnes, "Block-diagonal and LT codes for distributed computing with straggling servers," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 1739–1753, Mar. 2019.

[6] A. Severinson, A. Graell i Amat, E. Rosnes, F. Lázaro, and G. Liva, "A droplet approach based on Raptor codes for distributed computing with straggling servers," in *Proc. 10th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Hong Kong, China, Dec. 2018.

[7] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Proc. Neural Inf. Process. Syst. (NIPS)*, Long Beach, CA, USA, Dec. 2017, pp. 4403–4413.

[8] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Trans. Inf. Theory*, vol. 65, no. 7, pp. 4227–4242, Jul. 2019.

[9] S. Dutta, V. Cadambe, and P. Grover, ""Short-Dot": Computing large linear transforms distributedly using coded short dot products," *IEEE Trans. Inf. Theory*, vol. 65, no. 10, pp. 6171–6193, Oct. 2019.

[10] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Trans. Inf. Theory*, vol. 66, no. 1, pp. 278–301, Jan. 2020.

[11] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Sydney, Australia, Aug. 2017, pp. 3368–3376.

[12] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Proc. Neural Inf. Process. Syst. (NIPS)*, Long Beach, CA, USA, Dec. 2017, pp. 5440–5448.

[13] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, pp. 58:1–58:40, Dec. 2019.

[14] K. Li, M. Tao, and Z. Chen, "Exploiting computation replication for mobile edge computing: A fundamental computation-communication tradeoff study," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4563–4578, Jul. 2020.

[15] K. Li, M. Tao, and Z. Chen, "A computation-communication tradeoff study for mobile edge computing networks," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Paris, France, Jul. 2019, pp. 2639–2643.

[16] K. Li, M. Tao, J. Zhang, and O. Simeone, "Multi-cell mobile edge coded computing: Trading communication and computing for distributed matrix multiplication," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Los Angeles, CA, USA, Jun. 2020, pp. 215–220.

[17] J. Zhang and O. Simeone, "On model coding for distributed inference and transmission in mobile edge computing systems," *IEEE Commun. Lett.*, vol. 23, no. 6, pp. 1065–1068, Jun. 2019.

[18] K. Li, M. Tao, J. Zhang, and O. Simeone, "Coded computing and cooperative transmission for wireless distributed matrix multiplication," *IEEE Trans. Commun.*, vol. 69, no. 4, pp. 2224–2239, Apr. 2021.

[19] A. Frigård, S. Kumar, E. Rosnes, and A. Graell i Amat, "Low-latency distributed inference at the network edge using rateless codes," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, Berlin, Germany, Sep. 2021.

[20] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure coded computing using secret sharing via staircase codes," *IEEE Trans. Commun.*, vol. 68, no. 8, pp. 4609–4619, Aug. 2020.

[21] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. El Rouayheb, and H. Seferoglu, "Prac: Private and rateless adaptive coded computation at the edge," in *Proc. SPIE Defense + Commercial Sensing*, Baltimore, MD, USA, May 2019.

[22] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. Int. Conf. Artificial Intell. Stats. (AISTATS)*, Naha, Japan, Apr. 2019, pp. 1215–1225.

[23] H. Yang and J. Lee, "Secure distributed computing with straggling servers using polynomial codes," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 1, pp. 141–150, Jan. 2019.

[24] A. Felfernig, S. Polat-Erdeniz, C. Uran, S. Reiterer, M. Atas, T. N. T. Tran, P. Azzoni, C. Kiraly, and K. Dolui, "An overview of recommender systems in the internet of things," *J. Intell. Inf. Syst.*, vol. 52, no. 2, pp. 285–309, Apr. 2019.

[25] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[26] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[27] J. Zhang and O. Simeone, "Fundamental limits of cloud and cache-aided interference management with multi-antenna edge nodes," *IEEE Trans. Inf. Theory*, vol. 65, no. 8, pp. 5197–5214, Aug. 2019.

[28] N. Naderializadeh, M. A. Maddah-Ali, and A. S. Avestimehr, "Fundamental limits of cache-aided interference management," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Barcelona, Spain, Jul. 2016, pp. 2044–2048.

[29] S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in *Proc. 48th Annual Allerton Conf. Commun., Control, Comput.*, Monticello, IL, USA, Sep. 2010, pp. 1510–1517.

[30] G. Garrammone, "On decoding complexity of Reed-Solomon codes on the packet erasure channel," *IEEE Commun. Lett.*, vol. 17, no. 4, pp. 773–776, Apr. 2013.

[31] R. Yavne, "An economical method for calculating the discrete Fourier transform," in *Proc. Joint Comput. Conf.*, San Francisco, CA, USA, Dec. 1968, pp. 115–125.

[32] D. R. Hughes and F. Piper, *Design Theory*. Cambridge University Press, U.K., 1985.

Paper II

CodedPaddedFL and CodedSecAgg: Straggler Mitigation and Secure Aggregation in Federated Learning

Reent Schlegel, Siddhartha Kumar, Eirik Rosnes, and Alexandre Graell i Amat

Submitted to IEEE Transactions on Communications.

Parts of this paper were presented at the IEEE International Conference on Communications (ICC), Seoul, South Korea, May 2022, and at the European Signal Processing Conference (EUSIPCO), Belgrade, Serbia, August/September 2022.

The layout has been revised.

Abstract

We present two novel federated learning (FL) schemes that mitigate the effect of straggling devices by introducing redundancy on the devices' data across the network. Compared to other schemes in the literature, which deal with stragglers or device dropouts by ignoring their contribution, the proposed schemes do not suffer from the client drift problem. The first scheme, CodedPaddedFL, mitigates the effect of stragglers while retaining the privacy level of conventional FL. It combines one-time padding for user data privacy with gradient codes to yield straggler resiliency. The second scheme, CodedSecAgg, provides straggler resiliency and robustness against model inversion attacks and is based on Shamir's secret sharing. We apply CodedPaddedFL and CodedSecAgg to a classification problem. For a scenario with 120 devices, CodedPaddedFL achieves a speed-up factor of 18 for an accuracy of 95% on the MNIST dataset compared to conventional FL. Furthermore, it yields similar performance in terms of latency compared to a recently proposed scheme by Prakash *et al.* without the shortcoming of additional leakage of private data. CodedSecAgg outperforms the state-of-the-art secure aggregation scheme LightSecAgg by a speed-up factor of 6.6–18.7 for the MNIST dataset for an accuracy of 95%.

1 Introduction

Federated learning (FL) [1–3] is a distributed learning paradigm that trains an algorithm across multiple devices without exchanging the training data directly, thus limiting the privacy leakage and reducing the communication load. More precisely, FL enables multiple devices to collaboratively learn a global model under the coordination of a central server. At each epoch, the devices train a local model on their local data and send the locally-trained models to the central server. The central server aggregates the local models to update the global model, which is sent to the devices for the next epoch of the training. FL has been used in real-world applications, e.g., for medical data [4], text predictions on mobile devices [5], or by Apple to personalize Siri.

Training over many heterogeneous devices can be detrimental to the overall latency due to the effect of so-called stragglers, i.e., devices that take exceptionally long to finish their tasks due to random phenomena such as processes running in the background and memory access. Dropouts, which can be seen as an extreme case of straggling, may also occur. One of the most common ways to address the straggling/dropout problem in FL is to ignore the result of the slowest devices, such as in federated averaging [1]. However, while this approach has only a small impact on the training accuracy when the data is homogeneous across devices, ignoring updates from the slowest devices can lead to the *client drift* problem when the data is not identically distributed across devices[6, 7]—the global model will

tend toward local solutions of the fastest devices, which impairs the overall accuracy of the scheme. In [7–12], asynchronous schemes have been proposed for straggler mitigation with non-identically distributed data in which the central server utilizes stale gradients from straggling devices. However, these schemes do not in general converge to the global optimum[7].

FL is also prone to model inversion attacks [13, 14], which allow the central server to infer information about the local datasets through the local gradients collected in each epoch. To prevent such attacks and preserve users' data privacy, secure aggregation protocols have been proposed [15–27] in which the central server only obtains the sum of all the local model updates instead of the local updates directly. The schemes in [15–27] provide security against inversion attacks by hiding devices' local models via masking. The masks have an additive structure so that they can be removed when aggregated at the central server. To provide resiliency against stragglers/dropouts, secret sharing of the random seeds that generate the masks between the devices is performed, so that the central server can cancel the masks belonging to dropped devices. Among these schemes, LightSecAgg [20] is one of the most efficient. The schemes [15–25] ignore the contribution of straggling and dropped devices. However, ignoring straggling (or dropped) devices makes these schemes sensitive to the client drift problem. The schemes in [26, 27] are asynchronous straggler-resilient schemes that do not in general converge to the global optimum.

The straggler problem has been addressed in the neighboring area of distributed computing using tools from coding theory. The key idea is to introduce redundancy on the data via an erasure correcting code before distributing it to the servers so that the computations of a subset of the servers are sufficient to complete the global computation, i.e., the computations of straggling servers can be ignored without loss of information. Coded distributed computing has been proposed for matrix-vector and matrix-matrix multiplication [28–34], distributed gradient descent [35], and distributed optimization [36].

Coding for straggler mitigation has also been proposed for edge computing [37–39] and FL[40]. The scheme in [40] lets each device generate parity data on its local data and share it with the central server. This allows the central server to recover part of the information corresponding to the local gradients of the straggling devices without waiting for their result in every epoch. However, sharing parity data with the central server leaks information and hence the scheme provides a lower privacy level than conventional FL.

In this paper, borrowing tools from coded distributed computing and edge computing, we propose two novel FL schemes, referred to as CodedPaddedFL and CodedSecAgg, that provide resiliency against straggling devices (and hence dropouts) by introducing redundancy on the devices' local data. Both schemes can be divided into two phases. In the first phase, the devices share an encoded version of their data with other devices. In the second phase, the devices and the central server iteratively and collaboratively train a global model. The proposed schemes achieve significantly lower training latency than state-of-the-art schemes.

Our main contributions are summarized as follows.

- We present CodedPaddedFL, an FL scheme that provides resiliency against straggling devices while retaining the same level of privacy as conventional FL. CodedPaddedFL combines one-time padding to yield privacy with gradient codes [35] to provide straggler resilience. Compared to the recent scheme in [40], which also exploits erasure correcting codes to yield straggler resiliency, the proposed scheme does not leak additional information.

- We present CodedSecAgg, a secure aggregation scheme that provides straggler resiliency by introducing redundancy on the devices' local data via Shamir's secret sharing. CodedSecAgg provides information-theoretic security against model inversion up to a given number of colluding malicious agents (including the central server). CodedPaddedFL and CodedSecAgg provide convergence to the true global optimum and hence do not suffer from the client drift phenomenon.

- For both schemes, we introduce a strategy for grouping the devices that significantly reduces the initial latency due to the sharing of the data as well as the decoding complexity at the central server at the expense of a slightly reduced straggler mitigation capability.

- Neither one-time padding nor secret sharing can be applied to real-valued data. To circumvent this problem, the proposed schemes are based on a fixed-point arithmetic representation of the real data and subsequently fixed-point arithmetic operations.

To the best of our knowledge, our work is the first to apply coding ideas to mitigate the effect of stragglers in FL without leaking additional information.

The proposed schemes are tailored to linear regression.[1] However, they can also be applied to nonlinear problems via kernel embedding. We apply CodedPaddedFL and CodedSecAgg to a classification problem on the MNIST[45] and Fashion-MNIST[46] datasets. For a scenario with 120 devices, CodedPaddedFL achieves a speed-up factor of 18 on the MNIST dataset for an accuracy of 95% compared to conventional FL, while it shows similar performance in terms of latency compared to the scheme in [40] without leaking additional data. CodedSecAgg achieves a speed-up factor of 6.6 for 60 colluding agents and up to 18.7 for a single malicious agent compared to LightSecAgg for an accuracy of 95% on the MNIST dataset. Our numerical results include the impact of the decoding in the overall latency, which is often neglected in the literature (thus making comparisons unfair as the decoding complexity may have a significant impact on the global latency [31]).

[1] Linear regression plays an important role for example to obtain trend lines, and linear models have gained more attention in the literature recently in light of the explainability of machine learning models, see, e.g., [41, 42], albeit linear models not being exclusive to explainability [43]. For more information on the importance of linear regression, see, e.g., [44].

2 Preliminaries

2.1 Notation

We use uppercase and lowercase bold letters for matrices and vectors, respectively, italics for sets, and sans-serif letters for random variables, e.g., $\boldsymbol{X}$, $\boldsymbol{x}$, $\mathcal{X}$, and X represent a matrix, a vector, a set, and a random variable, respectively. An exception to this rule is $\boldsymbol{\epsilon}$, which will denote a matrix. Vectors are represented as row vectors throughout the paper. For natural numbers c and d, $\mathbf{1}_{c\times d}$ denotes an all-one matrix of size $c \times d$. The transpose of a matrix $\boldsymbol{X}$ is denoted as $\boldsymbol{X}^\top$. The support of a vector $\boldsymbol{x}$ is denoted by $\mathrm{supp}(\boldsymbol{x})$, while the gradient of a function $f(\boldsymbol{X})$ with respect to $\boldsymbol{X}$ is denoted by $\nabla_{\boldsymbol{X}} f(\boldsymbol{X})$. Furthermore, we represent the Euclidean norm of a vector $\boldsymbol{x}$ by $\|\boldsymbol{x}\|$, while the Frobenius norm of a matrix $\boldsymbol{X}$ is denoted by $\|\boldsymbol{X}\|_{\mathrm{F}}$. Given integers $a, b \in \mathbb{Z}$, $a < b$, we define $[a, b] \triangleq \{a, \ldots, b\}$, where $\mathbb{Z}$ is the set of integers, and $[a] \triangleq \{1, \ldots, a\}$ for a positive integer a. Additionally, we use $(a)_b$ as a shorthand notation for a mod b. For a real number e, $\lfloor e \rfloor$ is the largest integer less than or equal to e and $\lceil e \rceil$ is the smallest integer larger than or equal to e. The expectation of a random variable is denoted by $\mathbb{E}[]$, and we write $\sim \mathrm{geo}(1-p)$ to denote that follows a geometric distribution with failure probability p. $\mathsf{I}(\cdot;\cdot)$ denotes the mutual information and $\mathsf{H}(\cdot|\cdot)$ the conditional entropy.

2.2 Fixed-Point Numbers

Fixed-point numbers are rational numbers with a fixed-length integer part and a fixed-length fractional part. A fixed-point number with length ℓ bits and resolution f bits can be seen as an integer from $\mathbb{Z}_{\langle \ell \rangle} = [-2^{\ell-1}, 2^{\ell-1} - 1]$ scaled by 2^{-f}. In particular, for fixed-point number $\tilde{x}$ it holds that $\tilde{x} = \bar{x} \cdot 2^{-f}$ for some $\bar{x} \in \mathbb{Z}_{\langle \ell \rangle}$. We define the set of all fixed-point numbers with length ℓ and resolution f as $\mathbb{Q}_{\langle \ell, f \rangle} \triangleq \{\tilde{x} = \bar{x}2^{-f}, \bar{x} \in \mathbb{Z}_{\langle \ell \rangle}\}$. The set $\mathbb{Q}_{\langle \ell, f \rangle}$ is used to represent real numbers in the interval between $-2^{\ell-f-1}$ and $2^{\ell-f-1}$ with a finite amount of, i.e. ℓ, bits.

2.3 Cyclic Gradient Codes

Gradient codes[35] are a class of codes that have been suggested for straggler mitigation in distributed learning and work as follows. A central server encodes partitions of training data via a gradient code. These coded partitions are assigned to servers which perform gradient computations on the assigned coded data. The central server is then able to decode the sum of the gradients of all partitions by contacting only a subset of the servers. In particular, a gradient code that can tolerate $\beta - 1$ stragglers in a scenario with γ servers and γ partitions encodes γ partitions into γ codewords, one for each server, such that a linear combination of any $\gamma - \beta + 1$ codewords yields the sum of all gradients of all partitions. We will refer to such a code as a (β, γ) gradient code. A (β, γ) gradient code over $\mathbb{Q}_{\langle \ell, f \rangle}$ consists of an encoding matrix $\boldsymbol{B} \in \mathbb{Q}_{\langle \ell, f \rangle}^{\gamma \times \gamma}$ and a decoding matrix $\boldsymbol{A} \in \mathbb{Q}_{\langle \ell, f \rangle}^{S \times \gamma}$, where S is the number of straggling patterns the central server can decode. The encoding matrix $\boldsymbol{B}$ has a cyclic structure and the support of each row is of size β, while

the the support of each row of the decoding matrix $\boldsymbol{A}$ is of size $\gamma - \beta + 1$. The support of the i-th row of $\boldsymbol{B}$ dictates which partitions are included in the codeword at server i, and the entries of the i-row are the coefficients of the linear combination of the corresponding partitions at server i. Let $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_\gamma$ be the gradients on partition $1, \ldots, \gamma$. Then, the gradient computed by server i is given by the i-th row of $\boldsymbol{B}\left(\boldsymbol{g}_1^\top, \ldots, \boldsymbol{g}_\gamma^\top\right)^\top$. The central server waits for the gradients of the $\gamma - \beta + 1$ fastest servers to decode. Let $\mathcal{A}$ be the index set of these fastest devices. The central server picks the row of $\boldsymbol{A}$ with support $\mathcal{A}$ and applies the linear combination given by this row on the received gradients. In order for the central server to receive $\sum_i \boldsymbol{g}_i$, the requirements on $\boldsymbol{A}$ and $\boldsymbol{B}$ are

$$\boldsymbol{A}\boldsymbol{B} = \boldsymbol{1}_{S\times\gamma}\,. \tag{II.11}$$

The construction of $\boldsymbol{A}$ and $\boldsymbol{B}$ can be found in [35, Alg. 1] and [35, Alg. 2], respectively.

2.4 Shamir's Secret Sharing Scheme

Shamir's secret sharing scheme (SSS)[47] over some field $\mathbb{F}$ with parameters (n, k) encodes a secret $x \in \mathbb{F}$ into n shares $s_1, \ldots, s_n$ such that the mutual information between x and any set of less than k shares is zero, while any set of k or more shares contain sufficient information to reconstruct the secret x. More precisely, for any $\mathcal{I} \subset \{s_1, \ldots, s_n\}$ with $|\mathcal{I}| < k$ and any $\mathcal{J} \subseteq \{s_1, \ldots, s_n\}$ with $|\mathcal{J}| \geq k$, we have $\mathsf{I}(x; \mathcal{I}) = 0$ and $\mathsf{H}(x|\mathcal{J}) = 0$.

Shamir's SSS achieves these two properties by encoding x together with $k-1$ independent and uniformly random samples $\mathsf{r}_1, \ldots, \mathsf{r}_{k-1}$ using a nonsystematic (n, k) Reed-Solomon code. As a result, any subset of Reed-Solomon encoded symbols, i.e., shares, of size less than k is independently and uniformly distributed. This means that these shares do not reveal any information about x, i.e., $\mathsf{I}(x; \mathcal{I}) = 0$. On the other hand, the maximum distance separable property of Reed-Solomon codes guarantees that any k coded symbols are sufficient to recover the initial message, i.e., $\mathsf{H}(x|\mathcal{J}) = 0$, where $\mathcal{J}$ denotes the set of the k coded symbols.

3 System Model

In this paper, we consider a network of n devices and a central server. Each device i owns local data $\mathcal{D}_i = \left\{(\boldsymbol{x}_j^{(i)}, \boldsymbol{y}_j^{(i)}) \mid j \in [n_i]\right\}$ consisting of n_i points with feature vectors $\boldsymbol{x}_j^{(i)}$ and labels $\boldsymbol{y}_j^{(i)}$. The devices wish to collaboratively train a global linear model $\boldsymbol{\theta}$ with the help of the central server on everyone's data, consisting of $m = \sum_i n_i$ points in total. The model $\boldsymbol{\theta}$ can be used to predict a label $\boldsymbol{y}$ corresponding to a given feature vector $\boldsymbol{x}$ as $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{\theta}$. Our proposed schemes rely on one-time padding and secret sharing, both of which can not be applied on real-valued data. To circumvent this shortcoming we use a fixed-point representation of the data. In particular, we assume $\boldsymbol{x}_j^{(i)} \in \mathbb{Q}_{\langle \ell, f\rangle}^d$ and $\boldsymbol{y}_j^{(i)} \in \mathbb{Q}_{\langle \ell, f\rangle}^c$, where d is the size of the feature space and c the dimension of the label. Note that practical systems often

operate in fixed-point representation, hence our schemes do not incur in a limiting assumption.

We represent the data in matrix form as

$$\boldsymbol{X}^{(i)} = \left(\boldsymbol{x}_1^{(i)\top}, \dots, \boldsymbol{x}_{n_i}^{(i)\top}\right)^\top \text{ and } \boldsymbol{Y}^{(i)} = \left(\boldsymbol{y}_1^{(i)\top}, \dots, \boldsymbol{y}_{n_i}^{(i)\top}\right)^\top .$$

The devices try to infer the global model $\boldsymbol{\Theta}$ using federated gradient descent, which we describe next.

3.1 Federated Gradient Descent

For convenience, we collect the whole data (consisting of m data points) in matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ as

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_m \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}^{(1)} \\ \vdots \\ \boldsymbol{X}^{(n)} \end{pmatrix} \text{ and } \boldsymbol{Y} = \begin{pmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_m \end{pmatrix} = \begin{pmatrix} \boldsymbol{Y}^{(1)} \\ \vdots \\ \boldsymbol{Y}^{(n)} \end{pmatrix},$$

where $\boldsymbol{X}$ is of size $m \times d$ and $\boldsymbol{Y}$ of size $m \times c$. The global model $\boldsymbol{\Theta}$ can be found as the solution of the following minimization problem:

$$\boldsymbol{\Theta} = \arg\min_{\boldsymbol{\Theta}'} \; f(\boldsymbol{\Theta}') ,$$

where $f(\boldsymbol{\Theta})$ is the *global* loss function

$$f(\boldsymbol{\Theta}) \triangleq \frac{1}{2m} \sum_{l=1}^{m} \left\| \boldsymbol{x}_l \boldsymbol{\Theta} - \boldsymbol{y}_l \right\|^2 + \frac{\lambda}{2} \left\| \boldsymbol{\Theta} \right\|_{\mathrm{F}}^2 , \tag{II.12}$$

where λ is the regularization parameter.

Let

$$f_i(\boldsymbol{\Theta}) = \frac{1}{2n_i} \sum_{j=1}^{n_i} \| \boldsymbol{x}_j^{(i)} \boldsymbol{\Theta} - \boldsymbol{y}_j^{(i)} \|^2$$

be the *local* loss function at device i. We can then write the global loss function in (II.12) as

$$f(\boldsymbol{\Theta}) = \sum_{i=1}^{n} \frac{n_i}{m} f_i(\boldsymbol{\Theta}) + \frac{\lambda}{2} \left\| \boldsymbol{\Theta} \right\|_{\mathrm{F}}^2 .$$

In federated gradient descent, the model $\boldsymbol{\Theta}$ is trained iteratively over multiple epochs on the local data at each device. At each epoch, the devices compute the gradient on the local loss function of the current model and send it to the central server. The central server then aggregates the local gradients to obtain a global gradient which is used to update the model. More precisely, during the e-th epoch, device i computes the gradient

$$\boldsymbol{G}_i^{(e)} = n_i \nabla_{\boldsymbol{\Theta}} f_i(\boldsymbol{\Theta}^{(e)}) = \boldsymbol{X}^{(i)\top} \boldsymbol{X}^{(i)} \boldsymbol{\Theta}^{(e)} - \boldsymbol{X}^{(i)\top} \boldsymbol{Y}^{(i)} , \tag{II.13}$$

where $\boldsymbol{\theta}^{(e)}$ denotes the current model estimate. Upon reception of the gradients, the central server aggregates them as $\boldsymbol{G}^{(e)} = \sum_i \boldsymbol{G}_i^{(e)}$ to update the model according to

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^{(e)}) = \frac{1}{m}\boldsymbol{G}^{(e)} + \lambda\boldsymbol{\theta}^{(e)}\,, \tag{II.14}$$

$$\boldsymbol{\theta}^{(e+1)} = \boldsymbol{\theta}^{(e)} - \mu\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^{(e)})\,, \tag{II.15}$$

where μ is the learning rate. The updated model $\boldsymbol{\theta}^{(e+1)}$ is then sent back to the devices, and (II.13) to (II.15) are iterated E times until convergence, i.e., until $\boldsymbol{\theta}^{(E+1)} \approx \boldsymbol{\theta}^{(E)}$.

3.2 Computation and Communication Latency

We model the computation times of the devices as random variables with a shifted exponential distribution, as is common in the literature [38]. This means that the computation times comprise a deterministic time corresponding to the time a device takes to finish a computation in its processing unit and a random setup time due to unforseen delays such as memory access and other tasks running in the background. Let $\mathsf{T}_i^{\mathsf{comp}}$ be the time it takes device i to perform ρ_i multiply and accumulate (MAC) operations. We then have

$$\mathsf{T}_i^{\mathsf{comp}} = \frac{\rho_i}{\tau_i} + {}_i\,,$$

with τ_i being the deterministic number of MAC operations device i performs per second and ${}_i$ the random exponentially distributed setup time with $\mathbb{E}[{}_i] = 1/\eta_i$.

The devices communicate with the central server through a secured, i.e., authenticated and encrypted, wireless link. This communication link is unreliable and may fail. In case a packet is lost, the sender retransmits until a successful transmission occurs. Let $\mathsf{N}_i^{\mathsf{u}} \sim \text{geo}\,(1-p_i)$ and $\mathsf{N}_i^{\mathsf{d}} \sim \text{geo}\,(1-p_i)$ be the number of tries until device i successfully uploads and downloads a packet to or from the central server, and let γ^{u} and γ^{d} be the transmission rates in the upload and download. Then, the time it takes device i to upload or download b bits is

$$\mathsf{T}_i^{\mathsf{u}} = \frac{\mathsf{N}_i^{\mathsf{u}}}{\gamma^{\mathsf{u}}}b \quad \text{and} \quad \mathsf{T}_i^{\mathsf{d}} = \frac{\mathsf{N}_i^{\mathsf{d}}}{\gamma^{\mathsf{d}}}b\,,$$

respectively. Furthermore, we assume that the devices feature full-duplex transmission capabilities, as per the LTE Cat 1 standard for Internet of Things (IoT) devices, and have access to orthogonal channels to the central server. This means that devices can simultaneously transmit and receive messages to and from the central server without interference from the other devices.[2]

Last but not least, all device-to-device (D2D) communication is authenticated and encrypted and is routed through the central server. This enables efficient D2D communication, because the routing through the central server guarantees that

[2]The proposed schemes apply directly to the half-duplex case as well. However, the full-duplex assumption allows to reduce the latency and simplifies its analysis.

any two devices in the network can communicate with each other even when they are spatially separated, and the authentication and encryption prohibit man-in-the-middle attacks and eavesdropping.

3.3 Threat Model and Goal

We assume a scenario where the central server and the devices are honest-but-curious. The goal of CodedPaddedFL is to provide straggler resiliency while achieving the same level of privacy as conventional FL, i.e., the central server does not gain additional information compared to conventional FL, and (colluding) devices do not gain any information on the data shared by other devices. For the secure aggregation scheme, CodedSecAgg, we assume that up to z agents (including the central server) may collude to infer information about the local datasets of other devices. The goal is to ensure device data privacy against the z colluding agents while providing straggler mitigation. Privacy in this setting means that malicious devices do not gain any information about the local datasets of other devices and that the central server only learns the aggregate of all local gradients to prevent a model inversion attack.

4 Privacy-Preserving Operations on Fixed-Point Numbers

CodedPaddedFL, introduced in the next section, is based on one-time padding to provide data privacy, while CodedSecAgg, introduced in Section 6, is based on Shamir's SSS. As mentioned before, neither one-time padding nor secret sharing can be applied over real-valued data. Hence, we resort to using a fixed-point representation of the data. In this section, we explain how to perform elementary operations on fixed-point numbers.

Using fixed-point representation of data for privacy-preserving computations was first introduced in [48] in the context of multi-party computation. The idea is to use the integer $\bar{x}$ to represent the fixed-point number $\tilde{x} = \bar{x} \cdot 2^{-f}$. To this end, the integer $\bar{x}$ is mapped into a finite field and can then be secretly shared with other devices to perform secure operations such as addition, multiplication, and division with other secretly shared values. In CodedPaddedFL and CodedSecAgg we will use a similar approach. However, we only need to multiply a known number with a padded number and not two padded numbers with each other, which significantly simplifies the operations.

Consider the fixed-point datatype $\mathbb{Q}_{\langle \ell, f \rangle}$ (see Section 2.2). Secure addition on $\mathbb{Q}_{\langle \ell, f \rangle}$ can be performed via simple integer addition with an additional modulo operation. Let $(\cdot)_{\mathbb{Z}_{\langle \ell \rangle}}$ be the map from the integers onto the set $\mathbb{Z}_{\langle \ell \rangle}$ given by the modulo operation. Furthermore, let $\tilde{a}, \tilde{b} \in \mathbb{Q}_{\langle \ell, f \rangle}$, with $\tilde{a} = \bar{a} 2^{-f}$ and $\tilde{b} = \bar{b} 2^{-f}$. For $\tilde{c} = \tilde{a} + \tilde{b}$, with $\tilde{c} = \bar{c} 2^{-f}$, we have $\bar{c} = (\bar{a} + \bar{b})_{\mathbb{Z}_{\langle \ell \rangle}}$.

Multiplication on $\mathbb{Q}_{\langle \ell, f \rangle}$ is performed via integer multiplication with scaling over the reals in order to retain the precision of the datatype and an additional modulo operation. For $\tilde{d} = \tilde{a} \cdot \tilde{b}$, with $\tilde{d} = \bar{d} 2^{-f}$, we have $\bar{d} = (\lfloor \bar{a} \cdot \bar{b} \cdot 2^{-f} \rceil)_{\mathbb{Z}_{\langle \ell \rangle}}$.

Proposition 1 (Perfect privacy). *Consider a secret $\tilde{x} \in \mathbb{Q}_{\langle \ell, f \rangle}$ and a one-time pad $\tilde{r} \in \mathbb{Q}_{\langle \ell, f \rangle}$ that is picked uniformly at random. Then, $\tilde{x} + \tilde{r}$ is uniformly distributed in $\mathbb{Q}_{\langle \ell, f \rangle}$, i.e., $\tilde{x} + \tilde{r}$ does not reveal any information about $\tilde{x}$.*

Proposition 1 is an application of a one-time pad, which was proven secure by Shannon in [49]. It follows that given that an adversary (having unbounded computational power) obtains the sum of the secret and the pad, $\tilde{x} + \tilde{r}$, and does not know the pad $\tilde{r}$, it cannot determine the secret $\tilde{x}$.

Proposition 2 (Retrieval). *Consider a public fixed-point number $\tilde{c} \in \mathbb{Q}_{\langle \ell, f \rangle}$, a secret $\tilde{x} \in \mathbb{Q}_{\langle \ell, f \rangle}$, and a one-time pad $\tilde{r} \in \mathbb{Q}_{\langle \ell, f \rangle}$ that is picked uniformly at random. Suppose we have the weighted sum $\tilde{c}(\tilde{x} + \tilde{r})$ and the one-time pad. Then, we can retrieve $\tilde{c}\tilde{x} = \tilde{c}(\tilde{x} + \tilde{r}) - \tilde{c}\tilde{r} + O(2^{-f})$.*

The above proposition tells us that, given $\tilde{c}$, $\tilde{c}(\tilde{x} + \tilde{r})$, and $\tilde{r}$, it is possible to obtain an approximation of $\tilde{c}\tilde{x}$. Moreover, if we choose a sufficiently large f, then we can retrieve $\tilde{c}\tilde{x}$ with negligible error.

5 Coded Federated Learning

In this section, we introduce our first proposed scheme, named CodedPaddedFL. To yield straggler mitigation, CodedPaddedFL is based on the use of gradient codes (see Section 2.3). More precisely, each device computes the gradient on a linear combination of the data of a subset of the devices. In contrast to distributed computing, however, where a user willing to perform a computation has all the data available, in an FL scenario the data is inherently distributed across devices and hence gradient codes cannot be applied directly. Thus, to enable the use of gradient codes, we first need to share data between devices. To preserve data privacy, in CodedPaddedFL, our scheme one-time pads the data prior to sharing it.

CodedPaddedFL comprises two phases. In the first phase, devices share data to enable the use of gradient codes. In the second phase, coded gradient descent is applied on the padded data.[3] In the following, we describe both phases.

5.1 Phase 1: Data Sharing

In the first phase of CodedPaddedFL, the devices share a one-time padded version of their data with other devices. We explain next how the devices pad their data.

Each device i generates a pair of uniformly random one-time pads $\mathsf{R}_i^{\mathrm{G}} \in \mathbb{Z}_{\langle \ell \rangle}^{d \times c}$ and $\mathsf{R}_i^{\mathrm{X}} \in \mathbb{Z}_{\langle \ell \rangle}^{d \times d}$, with $\mathsf{R}_i^{\mathrm{X}} = \mathsf{R}_i^{\mathrm{X}\mathsf{T}}$. Then, device i sends these one-time pads to the central server. Furthermore, using the one-time pads and its data, device i computes

$$\boldsymbol{\Psi}_i = \boldsymbol{G}_i^{(1)} + \mathsf{R}_i^{\mathrm{G}}, \tag{II.16}$$

$$\boldsymbol{\Phi}_i = \boldsymbol{X}^{(i)\mathsf{T}} \boldsymbol{X}^{(i)} + \mathsf{R}_i^{\mathrm{X}}, \tag{II.17}$$

[3] We remark that our proposed scheme deviates slightly from standard federated gradient descent as described in Section 3.1 by trading off a pre-computation of the data for more efficient computations at each epoch, as explained in Section 5.2.

where $\boldsymbol{G}_i^{(1)}$ is the gradient of device i in the first epoch (see (II.13)). Matrices $\boldsymbol{\Psi}_i$ and $\boldsymbol{\Phi}_i$ are one-time padded versions of the first gradient and the transformed data. As a result, the mutual information between them and the data at device i is zero. The reason for padding the gradient of the first epoch in (II.16) and the transformation of the dataset in (II.17) will become clear in Section 5.2.

Devices then share the padded matrices $\boldsymbol{\Psi}_i$ and $\boldsymbol{\Phi}_i$ with $\alpha - 1$ other devices to introduce redundancy in the network and enable coded gradient descent in the second phase. Particularly, as described in Section 5.2, each device computes the gradient on a linear combination of a subset of $\{\boldsymbol{\Psi}_1, \dots, \boldsymbol{\Psi}_n\}$ and $\{\boldsymbol{\Phi}_1, \dots, \boldsymbol{\Phi}_n\}$, where the linear combination is determined by an (α, n) gradient code. Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be the decoding matrix and the encoding matrix of the gradient code, respectively. Each row and column of $\boldsymbol{B}$ has exactly α nonzero elements. The support of row i determines the subset of $\{\boldsymbol{\Psi}_1, \dots, \boldsymbol{\Psi}_n\}$ and $\{\boldsymbol{\Phi}_1, \dots, \boldsymbol{\Phi}_n\}$ on which device i will compute the gradient. Correspondingly, the support of column j dictates the subset of devices with which device j has to share its padded data $\boldsymbol{\Psi}_j$ and $\boldsymbol{\Phi}_j$. The cyclic structure of $\boldsymbol{B}$ guarantees that each device will utilize its own data, which is why each device shares its data with only $\alpha - 1$ other devices while we have α nonzero elements in each column.

The sharing of data between devices is specified by an $\alpha \times n$ assignment matrix $\boldsymbol{\Omega}$ whose i-th column corresponds to the support of the i-th row of matrix $\boldsymbol{B}$. Matrix $\boldsymbol{\Omega}$ is given by

$$\boldsymbol{\Omega} = \begin{pmatrix} 1 & 2 & \cdots & n \\ 2 & 3 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ (\alpha-1)_D + 1 & (\alpha)_D + 1 & \cdots & (\alpha-2)_D + 1 \end{pmatrix}.$$

The entry at row i and column j of $\boldsymbol{\Omega}$, ω_{ij}, identifies a device sharing its padded data with device j, e.g., $\omega_{\cdot,a} = b$ means that device b shares its data with device a.

Example 3. *Consider $n = 3$ devices and $\alpha = 2$. We have the transmission matrix $\boldsymbol{\Omega} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$, where, for instance, $\omega_{21} = 2$ denotes that device 2 shares its padded gradient and data, $\boldsymbol{\Psi}_2$ and $\boldsymbol{\Phi}_2$, with device 1. The first row says that each device should share its data with itself, making communication superfluous, whereas the second row says that devices 2, 3, and 1 should share their padded gradients and data with devices 1, 2, and 3, respectively.*

After the sharing of the padded data, the devices locally encode the local data and the received data using the gradient code. Let $\{b_{i,j}\}$ be the entries of the encoding matrix $\boldsymbol{B}$. Device i then computes

$$\boldsymbol{C}_i = \left(b_{i,\omega_{1i}}, \dots, b_{i,\omega_{\alpha i}}\right)\left(\boldsymbol{\Psi}_{\omega_{1i}}^\top, \dots, \boldsymbol{\Psi}_{\omega_{\alpha i}}^\top\right)^\top, \tag{II.18}$$

$$\bar{\boldsymbol{C}}_i = \left(b_{i,\omega_{1i}}, \dots, b_{i,\omega_{\alpha i}}\right)\left(\boldsymbol{\Phi}_{\omega_{1i}}^\top, \dots, \boldsymbol{\Phi}_{\omega_{\alpha i}}^\top\right)^\top, \tag{II.19}$$

where (II.18) corresponds to the encoding, via the gradient code, of the padded gradient of device i at epoch 1 and the padded gradients (at epoch 1) received from the $\alpha - 1$ other devices, and (II.19) corresponds to the encoding of the padded data of device i as well as the padded data received from the other devices. This concludes the sharing phase of CodedPaddedFL.

5.2 Phase 2: Coded Gradient Descent

In the second phase of CodedPaddedFL, the devices and the central server collaboratively and iteratively train the global model $\boldsymbol{\theta}$. As the training is an iterative process, the model changes in each epoch. Let $\boldsymbol{\theta}^{(e)}$ be the model at epoch e. We can write $\boldsymbol{\theta}^{(e)}$ as

$$\boldsymbol{\theta}^{(e)} = \boldsymbol{\theta}^{(1)} + \boldsymbol{\epsilon}^{(e)}, \tag{II.20}$$

where $\boldsymbol{\epsilon}^{(e)}$ is an update matrix and $\boldsymbol{\theta}^{(1)}$ is the initial model estimate in the first epoch. In contrast to the standard approach in gradient descent, where the central server sends $\boldsymbol{\theta}^{(e)}$ to the devices in every epoch, we will use the update matrix $\boldsymbol{\epsilon}^{(e)}$ instead.

When device i receives $\boldsymbol{\epsilon}^{(e)}$, it computes the gradient $\tilde{\boldsymbol{G}}_i^{(e)}$ on the encoded data $\boldsymbol{C}_i$ and $\bar{\boldsymbol{C}}_i$. More precisely, in epoch e device i computes

$$\begin{aligned}
\tilde{\boldsymbol{G}}_i^{(e)} &= \boldsymbol{C}_i + \bar{\boldsymbol{C}}_i \boldsymbol{\epsilon}^{(e)} \\
&\overset{(a)}{=} \sum_{j=1}^{\alpha} b_{i,\omega_{ji}} \left(\boldsymbol{G}_{\omega_{ji}}^{(1)} + \mathsf{R}_{\omega_{ji}}^{\mathrm{G}} \right) + \sum_{j=1}^{\alpha} b_{i,\omega_{ji}} \left(\boldsymbol{X}^{(\omega_{ji})\top} \boldsymbol{X}^{(\omega_{ji})} + \mathsf{R}_{\omega_{ji}}^{\mathrm{X}} \right) \boldsymbol{\epsilon}^{(e)} \\
&\overset{(b)}{=} \sum_{j=1}^{\alpha} b_{i,\omega_{ji}} \left(\boldsymbol{G}_{\omega_{ji}}^{(1)} + \boldsymbol{X}^{(\omega_{ji})\top} \boldsymbol{X}^{(\omega_{ji})} \boldsymbol{\epsilon}^{(e)} \right) + \sum_{j=1}^{\alpha} b_{i,\omega_{ji}} \left(\mathsf{R}_{\omega_{ji}}^{\mathrm{G}} + \mathsf{R}_{\omega_{ji}}^{\mathrm{X}} \boldsymbol{\epsilon}^{(e)} \right) \\
&\overset{(c)}{=} \sum_{j=1}^{\alpha} b_{i,\omega_{ji}} \left(\boldsymbol{G}_{\omega_{ji}}^{(e)} + \mathsf{R}_{\omega_{ji}}^{\mathrm{X}} \boldsymbol{\epsilon}^{(e)} + \mathsf{R}_{\omega_{ji}}^{\mathrm{G}} \right),
\end{aligned} \tag{II.21}$$

where (a) follows (II.18) and (II.19) together with (II.16) and (II.17), (b) is a reordering, and (c) follows (II.13) and (II.20). Subsequently, device i sends the gradient $\tilde{\boldsymbol{G}}_i^{(e)}$ to the central server. The central server waits for the gradients from the $n - \alpha + 1$ fastest devices before it starts the decoding process, i.e., the central server ignores the results from the $\alpha - 1$ slowest devices, which guarantees resiliency against up to $\alpha - 1$ stragglers. The decoding is based on the decoding matrix $\boldsymbol{A}$. Let $\mathcal{A} \subseteq [n]$, with $|\mathcal{A}| = n - \alpha + 1$, be the set of the $n - \alpha + 1$ fastest devices. The central server, knowing all one-time pads and $\boldsymbol{\epsilon}^{(e)}$, removes the pads from $\tilde{\boldsymbol{G}}_i^{(e)}$, $\forall i \in \mathcal{A}$, and obtains $\boldsymbol{P}_i^{(e)} \triangleq \sum_{j=1}^{\alpha} b_{i,\omega_{ji}} \boldsymbol{G}_{\omega_{ji}}^{(e)}$. The next step is standard gradient code decoding. Let $\boldsymbol{a}_s = (a_{s,1}, \ldots, a_{s,n})$ be row s from $\boldsymbol{A}$ such that $\mathrm{supp}(\boldsymbol{a}_s) = \mathcal{A}$, i.e., row s is used to decode the straggling pattern $[n] \backslash \mathcal{A}$. Then,

$$\sum_{i \in \mathcal{A}} a_{s,i} \boldsymbol{P}_i^{(e)} \overset{(a)}{=} \boldsymbol{G}^{(e)} \overset{(b)}{=} m \big(\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^{(e)}) - \lambda \boldsymbol{\theta}^{(e)} \big), \tag{II.22}$$

where (a) follows the property of gradient codes in (II.11) and (b) follows (II.14). Lastly, $\boldsymbol{\theta}^{(e+1)}$ is obtained according to (II.15) for the next epoch.

The proposed CodedPaddedFL is schematized in Fig. II.7. It is easy to see that our scheme achieves the global optimum.

Proposition 3. *The proposed CodedPaddedFL with parameters (α, n) is resilient to $\alpha - 1$ stragglers, and achieves the global optimum, i.e., the optimal model obtained through gradient descent computed over the devices' datasets for linear regression.*

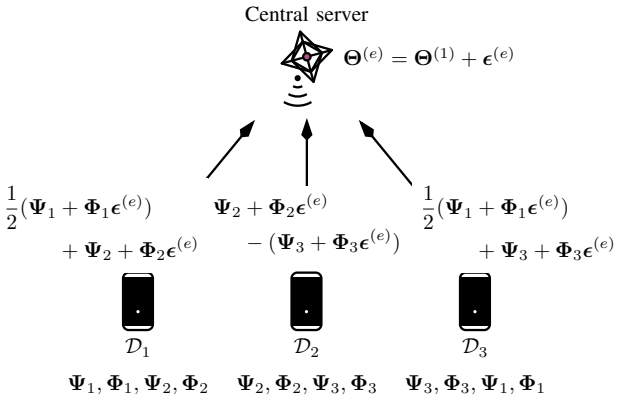


Figure II.7: An example showcasing the system model as well as an epoch of the proposed CodedPaddedFL. The system consists of $n = 3$ devices and a central server. The devices share $\boldsymbol{\Psi}_i$ and $\boldsymbol{\Phi}_i$. During the e-th epoch, the central server sends $\boldsymbol{\epsilon}^{(e)}$ to the devices. The devices compute coded gradients using an $(\alpha = 2, n)$ gradient code, and send them to the central server, which decodes them to compute the model update.

Proof: From (II.22), we see that during each epoch, e, the central server obtains

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^{(e)}) = \frac{1}{m}\boldsymbol{G}^{(e)} + \lambda\boldsymbol{\theta}^{(e)} = \frac{1}{m}\boldsymbol{X}^{\top}(\boldsymbol{X}\boldsymbol{\theta}^{(e)} - \boldsymbol{Y}) + \lambda\boldsymbol{\theta}^{(e)}$$

using the coded data obtained from the $n-\alpha+1$ fastest devices. It further obtains an updated linear model using (II.15), which is exactly the update rule for gradient descent. ∎

5.3 Communication Latency of the Data Sharing Phase

As mentioned in Section 3.2, we assume that devices are equipped with full-duplex technology and that simultaneous transmission between the n devices and the central server via orthogonal channels is possible. Thus, the sharing of data between devices according to $\boldsymbol{\Omega}$ requires $\alpha - 1$ successive transmissions consisting of upload and download. In particular, the first row of $\boldsymbol{\Omega}$ encompasses no transmission as each device already has access to its own data. For the other rows of $\boldsymbol{\Omega}$, the communication corresponding to the data sharing specified by any given row can be performed simultaneously, as the devices can communicate in full-duplex with the central server, and each device has a dedicated channel without interference from the other devices. As a result, all data sharing as defined by $\boldsymbol{\Omega}$ is completed after $\alpha - 1$ successive uploads and downloads. Considering Example 3, we can see that $\alpha - 1 = 1$ transmission is enough.

Remark 2. *Note that because both $\boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}$ and R_i^X are symmetric, $\boldsymbol{\Phi}_i$ is symmetric as well. As a result, device i only has to transmit the upper half of $\boldsymbol{\Phi}_i$ to the other devices.*

We assume the communication cost of transmitting the one-time pads $\mathsf{R}_i^{\mathsf{G}}$ and $\mathsf{R}_i^{\mathsf{X}}$ to the central server to be negligible as in practice the pads will be generated using a pseudorandom number generator such that it is sufficient to send the much smaller seed of the pseudorandom number generator instead of the whole one-time pads.

5.4 Complexity

We analyze the complexity of the two phases of CodedPaddedFL.

In the sharing phase, when $\alpha > 1$, device i has to upload $\boldsymbol{\Phi}_i$ and $\boldsymbol{\Psi}_i$ to the central server and download $\alpha - 1$ different $\boldsymbol{\Phi}_j$ and $\boldsymbol{\Psi}_j$ from other devices as given by the encoding matrix $\boldsymbol{B}$. As a result, the sharing comprises uploading $d\left(\frac{d+1}{2} + c\right)$ and downloading $(\alpha - 1)\, d\left(\frac{d+1}{2} + c\right)$ elements from $\mathbb{Q}_{\langle \ell, f \rangle}$. The encoding encompasses linearly combining α matrices two times (both $\boldsymbol{\Phi}_i$ and $\boldsymbol{\Psi}_i$). Therefore, each device has to perform $(\alpha - 1)\, d\left(\frac{d+1}{2} + c\right)$ MAC operations. In the case of $\alpha = 1$, no sharing of data and encoding takes place.

In the learning phase, devices have to compute a matrix multiplication with a subsequent matrix addition. This requires $d^2 c$ MAC operations. Subsequently, the devices transmit their model updates, consisting of dc elements from $\mathbb{Q}_{\langle \ell, f \rangle}$.

Remark 3. *During the learning phase, the complexity of CodedPaddedFL is equivalent to the complexity of conventional FL at the devices. The computations in* (II.21) *are as complex as in* (II.13) *given a pre-computation of* $\boldsymbol{X}^{(i)\mathsf{T}}\boldsymbol{X}^{(i)}$ *and* $\boldsymbol{X}^{(i)\mathsf{T}}\boldsymbol{Y}^{(i)}$ *and the model updates have the same dimensions.*

At each epoch, the central server has to decode the gradient code. Recall that $\boldsymbol{A}$ contains all the necessary information for decoding in the event of each straggling pattern the code is designed for. The central server simply has to pick the row of $\boldsymbol{A}$ with support equal to the indices of the nonstraggling devices and take a linear combination of the local results with coefficients from the given row of $\boldsymbol{A}$. This linear combination entails $(n - \alpha + 1)\, dc$ MAC operations. In contrast, for conventional FL, the master has to sum up n local gradients, which entails ndc additions.

The sizes of the coded data $\tilde{\boldsymbol{C}}_i$ and the coded gradient $\boldsymbol{C}_i$ are equal to the sizes of $\boldsymbol{X}^{(i)\mathsf{T}}\boldsymbol{X}^{(i)}$ and $\boldsymbol{G}_i^{(1)}$, respectively. The encoding comprises just a linear combination of multiple $\boldsymbol{X}^{(i)\mathsf{T}}\boldsymbol{X}^{(i)}$ and $\boldsymbol{G}_i^{(1)}$, which in turn have the same size as each $\boldsymbol{X}^{(i)\mathsf{T}}\boldsymbol{X}^{(i)}$ and $\boldsymbol{G}_i^{(1)}$. Each incoming $\boldsymbol{X}^{(i)\mathsf{T}}\boldsymbol{X}^{(i)}$ and $\boldsymbol{G}_i^{(1)}$ at the devices can directly be scaled and added to the existing intermediate results to avoid a buffering of multiple $\boldsymbol{X}^{(i)\mathsf{T}}\boldsymbol{X}^{(i)}$ and $\boldsymbol{G}_i^{(1)}$. Furthermore, when $n_i > d$, which is usually the case, $\boldsymbol{X}^{(i)\mathsf{T}}\boldsymbol{X}^{(i)}$ requires less storage space than $\boldsymbol{X}^{(i)}$, and $\boldsymbol{G}_i^{(1)}$ requires less space than $\boldsymbol{Y}^{(i)}$, which means that CodedPaddedFL might in fact require less storage space than conventional FL when training is performed directly on $\boldsymbol{X}^{(i)}$.

5.5 Grouping

To yield privacy, our proposed scheme entails a relatively high communication cost in the sharing phase and a decoding cost at the central server, which grow with increasing values of α. As we show in Section 8, values of α close to the maximum, i.e., n, yield the lowest overall latency, due to the strong straggler mitigation a high α facilitates. To reduce latency further, one should reduce α while retaining a high level of straggler mitigation. To achieve this, we partition the set of all devices into N smaller disjoint groups and locally apply CodedPaddedFL in each group. It is most efficient to distribute the devices among all N groups as equally as possible, as each group will experience the same latency. However, it is not necessary that N divides n.

The central server decodes the aggregated gradients from each group of devices first and obtains the global aggregate as the sum of the individual group aggregates.

Example 4. *Assume that there are $n = 25$ devices and the central server waits for the 10 fastest devices to finish their computation. This would result in $\alpha = 16$ as in CodedPaddedFL the central server has to wait for the $n - \alpha + 1$ fastest devices. By grouping the devices into $N = 5$ groups of $n/N = 5$ devices each, $\alpha = 4$ would be sufficient as the central server has to wait for the $n/N - \alpha + 1 = 5 - 4 + 1 = 2$ fastest devices in each group, i.e., 10 devices in total.*

Note that in the previous example the 2 fastest devices in each group are not necessarily among the 10 fastest devices globally. This means that the straggler mitigation capability of CodedPaddedFL with grouping may be lower than that of CodedPaddedFL with no grouping. As we will show numerically in Section 8, the trade-off of slightly reduced straggler mitigation for much lower values of α—thereby much lower initial communication load and lower decoding complexity at the central server—can reduce the overall latency.

6 Coded Secure Aggregation

In this section, we present a coding scheme, referred to as CodedSecAgg, for mitigating the effect of stragglers in FL that increases the privacy level of traditional FL schemes and CodedPaddedFL by preventing the central server from launching a model inversion attack. The higher level of privacy compared to CodedPaddedFL is achieved at the expense of a higher training time.

As with CodedPaddedFL, CodedSecAgg can be divided into two phases. First, the devices use Shamir's SSS (see Section 2.4) with parameters (n, k) to share their local data with other devices in the network. This introduces redundancy of the data which can be leveraged for straggler mitigation. At the same time, Shamir's SSS guarantees that any subset of less than k devices does not learn anything about the local datasets of other devices. In the second phase, the devices perform gradient descent on the SSS encoded data and send their results to the central server. The central server can decode the received results from any k devices to obtain the aggregated gradient, thereby providing resiliency against up to $n - k$ stragglers.

At the same time, the central server does not gain access to any local gradient and a model inversion attack is prevented.

6.1 Phase 1: Data Sharing

The devices use Shamir's SSS with parameters (n, k) to encode both $\boldsymbol{G}_i^{(1)}$ and $\boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}$ into n shares. Let $\{\mathsf{R}_{i,1}^{\mathrm{G}}, \dots, \mathsf{R}_{i,k-1}^{\mathrm{G}}\}$ and $\{\mathsf{R}_{i,1}^{\mathrm{X}}, \dots, \mathsf{R}_{i,k-1}^{\mathrm{X}}\}$ be two sets of $k-1$ independent and uniformly distributed matrices. Device i encodes $\boldsymbol{G}_i^{(1)}$ together with $\{\mathsf{R}_{i,1}^{\mathrm{G}}, \dots, \mathsf{R}_{i,k-1}^{\mathrm{G}}\}$ into n shares $\{\boldsymbol{\Psi}_i^{(1)}, \dots, \boldsymbol{\Psi}_i^{(n)}\}$ and $\boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}$ together with $\{\mathsf{R}_{i,1}^{\mathrm{X}}, \dots, \mathsf{R}_{i,k-1}^{\mathrm{X}}\}$ into n shares $\{\boldsymbol{\Phi}_i^{(1)}, \dots, \boldsymbol{\Phi}_i^{(n)}\}$ using a nonsystematic (n, k) Reed-Solomon code. Subsequently, each device sends one share of each encoding to each of the other $n-1$ devices. More precisely, device i sends $\boldsymbol{\Psi}_i^{(j)}$ and $\boldsymbol{\Phi}_i^{(j)}$ to device j.

Once the data sharing is completed, device i has $\{\boldsymbol{\Psi}_1^{(i)}, \dots, \boldsymbol{\Psi}_D^{(i)}\}$ and $\{\boldsymbol{\Phi}_1^{(i)}, \dots, \boldsymbol{\Phi}_D^{(i)}\}$. Device i then computes $\boldsymbol{\Psi}^{(i)} = \sum_{j=1}^{n} \boldsymbol{\Psi}_j^{(i)}$ and $\boldsymbol{\Phi}^{(i)} = \sum_{j=1}^{n} \boldsymbol{\Phi}_j^{(i)}$. It is easy to see that $\{\boldsymbol{\Psi}^{(1)}, \dots, \boldsymbol{\Psi}^{(n)}\}$ and $\{\boldsymbol{\Phi}^{(1)}, \dots, \boldsymbol{\Phi}^{(n)}\}$ correspond to applying Shamir's SSS with parameters (n, k) to $\{\sum_i \boldsymbol{G}_i^{(1)}, \sum_i \mathsf{R}_{i,1}^{\mathrm{G}}, \dots, \sum_i \mathsf{R}_{i,k-1}^{\mathrm{G}}\}$ and $\{\sum_i \boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}, \sum_i \mathsf{R}_{i,1}^{\mathrm{X}}, \dots, \sum_i \mathsf{R}_{i,k-1}^{\mathrm{X}}\}$. Hence, due to the linearity of Shamir's SSS, the devices now obtained successfully a secret share of $\boldsymbol{G}^{(1)}$ and $\boldsymbol{X}^{\top}\boldsymbol{X}$, the first aggregated global gradient and the global dataset. This concludes the first phase.

6.2 Phase 2: Securely Aggregated Gradient Descent

The second phase is an iterative learning phase, in which the devices continue to exploit the linearity of Shamir's SSS by computing the gradient updates on their shares $\boldsymbol{\Phi}^{(i)}$ and $\boldsymbol{\Psi}^{(i)}$. They thereby obtain a share of the new gradient in each epoch. More precisely, in each epoch e, the devices compute

$$\tilde{\boldsymbol{G}}_i^{(e)} = \boldsymbol{\Psi}^{(i)} + \boldsymbol{\Phi}^{(i)}\boldsymbol{\epsilon}^{(e)}. \tag{II.23}$$

In epoch e, the k fastest devices to finish their computation send their computed update $\tilde{\boldsymbol{G}}_i^{(e)}$ to the central server, which can decode the SSS to obtain the aggregated gradient $\boldsymbol{G}^{(e)}$ for that epoch. At the same time, the aggregated gradient is the only information the central server—and any set of less than k colluding devices—obtains. In the first phase, the local datasets are protected by the SSS from any inference, and in the second phase, only shares of the aggregated gradients are collected, which do not leak any information that the central server was not supposed to learn—the central server is supposed to learn the aggregated gradient in each epoch and there is no additional information the central server learns. We illustrate CodedSecAgg in Fig. II.8.

Remark 4. *In order to guarantee that the central server obtains the correct model update after decoding the SSS, the devices have to modify the multiplication of fixed-point numbers: in the SSS, we interpret the fixed-point numbers as integers from*

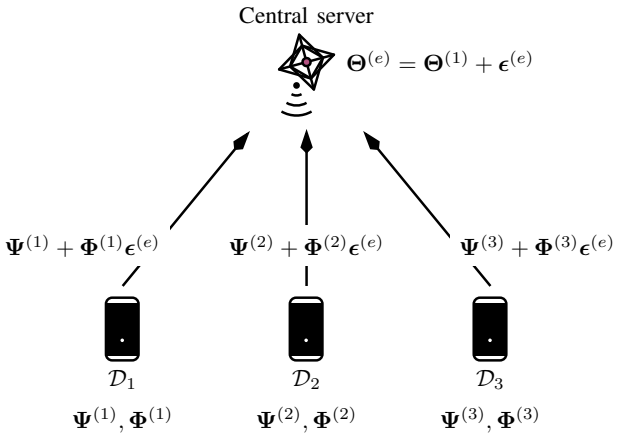


Figure II.8: An example showcasing an epoch of CodedSecAgg. The system consists of $n = 3$ devices and a central server. Each device has access to one share of the global dataset.

$\mathbb{Z}_{\langle \ell \rangle}$. As described in Section 4, multiplying two fixed-point numbers involves an integer multiplication with subsequent scaling to retain the precision of the datatype. However, it is not guaranteed that the decoding algorithm of the SSS will yield the desired result when the devices apply scaling after the integer multiplication. Whenever a wrap-around happens, i.e., the result of an integer operation exceeds $2^{\ell-1}$ or $-2^{\ell-1}$, which is expected to happen during the encoding of the SSS, the subsequent scaling distorts the arithmetic of the SSS. To circumvent this phenomenon and guarantee correct decoding of the global aggregate, we postpone the scaling and apply it after the decoding of the SSS at the central server. To this end, the range of integers we can represent has to be increased in accordance with the number of fractional bits used, i.e., the devices have to perform integer operations in $\mathbb{Z}_{\langle \ell+f \rangle}$, to guarantee that no overflows occur due to the postponed scaling. This entails an increase in computation and communication complexity of a factor $\frac{\ell+f}{\ell}$ compared to the case where no secret sharing is used. Furthermore, (II.23) *involves the addition of the result of a multiplication and a standalone matrix $\boldsymbol{\Psi}^{(i)}$. In order to perform a correct scaling after the decoding, we have to artificially multiply the standalone matrix $\boldsymbol{\Psi}^{(i)}$ with the identity matrix. This can be done efficiently by multiplying $\boldsymbol{G}_i^{(1)}$ with 2^f prior to encoding it into $\{\boldsymbol{\Psi}_i^{(1)}, \ldots, \boldsymbol{\Psi}_i^{(n)}\}$.*

Reed-Solomon codes are defined over finite fields. Thus, the operations in CodedSecAgg need to be performed over a finite field. To this end, for a fixed-point representation using ℓ bits (see Section 2.2), we consider a finite field of order $q > 2^{\ell+f}$, where q is a prime number. The mapping between the integers corresponding to the ℓ-bit fixed-point representation is done as follows: we map integers 0 to $2^{\ell+f-1} - 1$ to the first $2^{\ell+f-1}$ elements of the finite field and $-1 \mapsto q - 1$, $-2 \mapsto q - 2$, and so on.

By mapping the integers to finite field elements in this way, operations in the one domain directly translate to the other domain, i.e., additions and multiplications

of two elements from the finite field directly relate to additions and multiplications of the corresponding integers. The map from finite field elements back to the integers is straightforward and is given by the inverse of the map given above. But, since $q > 2^{\ell+f}$, there are some finite field elements with no corresponding integer in $\mathbb{Z}_{\langle \ell+f \rangle}$. However, we will only encounter these elements when the result of the performed operations exceeds $2^{\ell+f-1} - 1$ or $-2^{\ell+f-1}$, in which case we expect undesired behavior anyway. In other words, we have to choose ℓ large enough so that no overflows occur.

6.3 Complexity

The complexity analysis of CodedSecAgg is almost equivalent to that of CodedPaddedFL (see Section 5.4). In particular, the complexity of CodedSecAgg's learning phase at the devices is identical to that of CodedPaddedFL and conventional FL. We consider now the sharing phase. In the sharing phase, device i has to upload its $2\,(n-1)$ shares $\{\boldsymbol{\Psi}_i^{(j)} | j \in [n], j \neq i\}$ and $\{\boldsymbol{\Phi}_i^{(j)} | j \in [n], j \neq i\}$ and download the $2\,(n-1)$ shares $\{\boldsymbol{\Psi}_j^{(i)} | j \in [n], j \neq i\}$ and $\{\boldsymbol{\Phi}_j^{(i)} | j \in [n], j \neq i\}$. Furthermore, each device has to add n matrices twice. Therefore, each device uploads and downloads $(n-1)\,d\left(\frac{d+1}{2} + c\right)$ elements from the finite field and performs $(n-1)\,d\left(\frac{d+1}{2} + c\right)$ additions (MAC operations where one of the factors is set to 1).

At each epoch, the central server has to decode Shamir's SSS. This is equivalent to decoding the underlying (n, k) Reed-Solomon code. Because the central server has to only recover erasures and has to correct no errors, the coefficients for any straggling pattern can be pre-computed as was the case with the gradient code for CodedPaddedFL. Alternatively, the coefficients can be computed online via, e.g., polynomial interpolation. In any case, the central server has to compute a linear combination of k gradients which comprises kdc MAC operations.

As with CodedPaddedFL, the storage cost for CodedSecAgg is equal to the cost of storing $\boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}$ and $\boldsymbol{G}_i^{(1)}$, which in turn require less space than $\boldsymbol{X}^{(i)}$ and $\boldsymbol{Y}^{(i)}$ when $n_i > d$.

6.4 Grouping for Coded Secure Aggregation

For the proposed CodedSecAgg, the communication cost entailed by the sharing phase and the decoding cost at the central server increase with the number of devices n. Similar to CodedPaddedFL, we can reduce these costs while preserving straggler mitigation and secure aggregation by grouping the devices into groups and applying CodedSecAgg in each group. However, applying directly the above-described CodedSecAgg locally in each group would leak information about the data of subsets of devices. In particular, the central server would learn the aggregated gradients in each group instead of only the global gradient. To circumvent this problem, we introduce a hierarchical structure on the groups. Specifically, only one group, referred to as the master group, sends updates to the central server directly. This group collects the model updates of the other groups and aggregates

them before passing the global aggregate to the central server. To prevent a communication bottleneck at the master group, we avoid all devices sending updates directly to this group by dividing the communication into multiple hierarchical steps. At each step, we collect intermediate aggregates at fewer and fewer groups until all group updates are aggregated at the master group.

The proposed algorithm is as follows. We group devices into N disjoint groups. Contrary to CodedPaddedFL, where the groups may be of different size, here we require equally-sized groups, i.e., N divides n, as the inter-group communication requires each device in a group communicating with a unique device in another group and no two devices communicating with the same device. Furthermore, we require the number of devices in each group to be at least k. For notational purposes, we assign each group an identifier $j \in [N]$, and each device in a group is assigned an identifier $i \in [n/N]$ which is used to determine which shares each device receives from the other devices in its group in phase one of the algorithm. Let $\boldsymbol{G}_{j,i}^{(1)}$ be the first gradient of device i in group j and $\boldsymbol{X}^{(j,i)\top}\boldsymbol{X}^{(j,i)}$ its data. Similar to the above-described scheme, device i in group j applies Shamir's $(n/N, k)$ SSS on $\boldsymbol{G}_{j,i}^{(1)}$ and $\boldsymbol{X}^{(j,i)\top}\boldsymbol{X}^{(j,i)}$ to obtain n/N shares $\{\boldsymbol{\Psi}_{j,i}^{(1)}, \dots, \boldsymbol{\Psi}_{j,i}^{(n/N)}\}$ and n/N shares $\{\boldsymbol{\Phi}_{j,i}^{(1)}, \dots, \boldsymbol{\Phi}_{j,i}^{(n/N)}\}$, respectively. Device i sends shares $\boldsymbol{\Psi}_{j,i}^{(i')}$ and $\boldsymbol{\Phi}_{j,i}^{(i')}$ to all other devices $i' \in [n/N] \backslash i$ in the group. Device i' then computes $\boldsymbol{\Psi}_{j}^{(i')} = \sum_{i \in [n/N]} \boldsymbol{\Psi}_{j,i}^{(i')}$ and $\boldsymbol{\Phi}_{j}^{(i')} = \sum_{i \in [n/N]} \boldsymbol{\Phi}_{j,i}^{(i')}$.

Let $\tilde{\boldsymbol{G}}_{j,i}^{(e)} = \boldsymbol{\Psi}_{j}^{(i)} + \boldsymbol{\Phi}_{j}^{(i)} \boldsymbol{\epsilon}^{(e)}$, $j \in [N]$, $i \in [n/N]$, be the model update of device i in group j at epoch e equivalently to (II.23) and $\bar{\boldsymbol{G}}_{j}^{(e)}$ the aggregated gradient of group j in epoch e. Similar to the above-described CodedSecAgg scheme, $\{\tilde{\boldsymbol{G}}_{j,1}^{(e)}, \dots, \tilde{\boldsymbol{G}}_{j,n/N}^{(e)}\}$ is the result of applying Shamir's $(n/N, k)$ SSS on $\bar{\boldsymbol{G}}_{j}^{(e)}$. In order to prevent the central server from learning $\bar{\boldsymbol{G}}_{j}^{(e)}$ for any j, the key idea is to compute n/N shares of $\boldsymbol{G}^{(e)} = \sum_{j \in [N]} \bar{\boldsymbol{G}}_{j}^{(e)}$ in the master group. Devices in the master group can then send their shares of $\boldsymbol{G}^{(e)}$ to the central server, which can decode the SSS from any k shares to obtain $\boldsymbol{G}^{(e)}$. Note that we want to prevent the central server inferring any of the individual $\bar{\boldsymbol{G}}_{j}^{(e)}$. We can achieve this by letting device i in the master group compute $\sum_j \tilde{\boldsymbol{G}}_{j,i}^{(e)}$. As each $\tilde{\boldsymbol{G}}_{j,i}^{(e)}$ is one out of n/N shares of $\bar{\boldsymbol{G}}_{j}^{(e)}$, $\{\sum_j \tilde{\boldsymbol{G}}_{j,1}^{(e)}, \dots, \sum_j \tilde{\boldsymbol{G}}_{j,n/N}^{(e)}\}$ is the result of applying Shamir's SSS on $\sum_{j \in [N]} \bar{\boldsymbol{G}}_{j}^{(e)} = \boldsymbol{G}^{(e)}$. If the central server would aggregate $\sum_j \tilde{\boldsymbol{G}}_{j,i}^{(e)}$, share i of the global gradient $\boldsymbol{G}^{(e)}$, it could infer information about the local gradients in each group j, thereby violating the privacy guarantee of secure aggregation.

The model updates $\tilde{\boldsymbol{G}}_{j,i}^{(e)}$ are not sent directly to device i in the master group, to avoid a communication bottleneck in the master group. In particular, we divide the communication round into multiple steps. Assume for simplicity, and without loss of generality, that the master group is group one. We proceed as follows. In the first step, each device i in group $j \in \{j' \in [N] \mid j' \bmod 2 = 0\}$ sends $\tilde{\boldsymbol{G}}_{j,i}^{(e)}$ to device i in group $j - 1$, which adds its own share and the received one, i.e., the

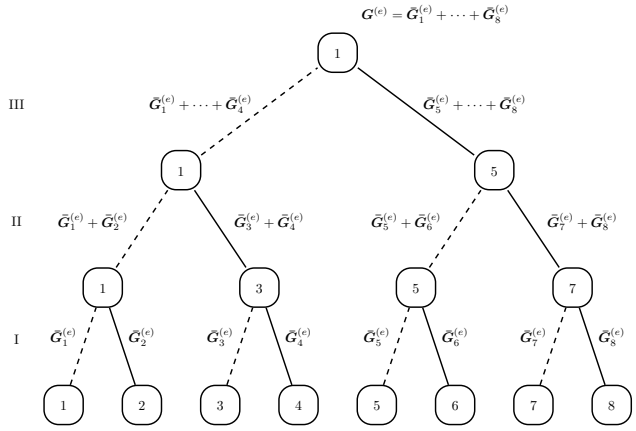


Figure II.9: An example of the inter-group communication for a network with $N = 8$ groups. Different layers correspond to the $\lceil \log_2(N) \rceil = 3$ communication steps, while the label of each node is the group identifier. A solid line between node i and node j represents a physical transmission from devices in group i to devices in group j, whereas dashed lines represent data already available at the end node.

devices in group $j-1$ obtain a share of $\bar{\boldsymbol{G}}_{j-1}^{(e)} + \bar{\boldsymbol{G}}_{j}^{(e)}$. In the second step, each device i in group $j \in \{j' \in [N] \mid j' \bmod 4 = 3\}$ sends $\tilde{G}_{j,i}^{(e)} + \tilde{G}_{j+1,i}^{(e)}$ to device i in group $j-2$ which again aggregates the received shares and its own. Devices in group $j-2$ now have obtained a share of $\bar{\boldsymbol{G}}_{j-2}^{(e)} + \bar{\boldsymbol{G}}_{j-1}^{(e)} + \bar{\boldsymbol{G}}_{j}^{(e)} + \bar{\boldsymbol{G}}_{j+1}^{(e)}$. Generally, in step s, each device i in group

$$j \in \{j' \in [N] \mid j' \bmod 2^s = 2^{s-1} + 1 \bmod 2^s\} \tag{II.24}$$

sends

$$\sum_{j'=j}^{\min\{j+2^{s-1}-1,N\}} \tilde{\boldsymbol{G}}_{j',i}^{(e)} \tag{II.25}$$

to device i in group $j - 2^{s-1}$. We continue this process until the devices in group one (the master group) have obtained shares of the global gradient $\boldsymbol{G}^{(e)} = \sum_j \bar{\boldsymbol{G}}_j^{(e)}$. In total, $\lceil \log_2(N) \rceil$ steps are needed to reach this goal. If the devices in each group would directly send their results to the devices in the first group, the communication latency would be linear in N instead of logarithmic, as $N-1$ results would have to be communicated to the devices in the first group sequentially.

We illustrate the inter-group communication with the following example.

Example 5. *Consider a network with $N = 8$ groups as depicted in Fig. II.9, where the groups are numbered 1 to 8 and represented by squares. In the first step, devices in group 2 send their shares of $\bar{\boldsymbol{G}}_2^{(e)}$ to devices in group 1, devices in group 4 their shares of $\bar{\boldsymbol{G}}_4^{(e)}$ to group 3, devices in group 6 their shares of $\bar{\boldsymbol{G}}_6^{(e)}$ to group 5, and devices in group 8 their shares of $\bar{\boldsymbol{G}}_8^{(e)}$ to group 7, which is illustrated by the solid*

lines. After the first step, each device in group 1 *has access to a share of* $\bar{\boldsymbol{G}}_1^{(e)} + \bar{\boldsymbol{G}}_2^{(e)}$, *devices in group* 3 *have shares of* $\bar{\boldsymbol{G}}_3^{(e)} + \bar{\boldsymbol{G}}_4^{(e)}$, *and so forth. In the second step, devices from group* 3 *send their shares of* $\bar{\boldsymbol{G}}_3^{(e)} + \bar{\boldsymbol{G}}_4^{(e)}$ *to devices in group* 1 *and devices from group* 7 *their shares of* $\bar{\boldsymbol{G}}_7^{(e)} + \bar{\boldsymbol{G}}_8^{(e)}$ *to group* 5. *In the last step, the devices in group* 5 *send their shares of* $\bar{\boldsymbol{G}}_5^{(e)} + \bar{\boldsymbol{G}}_6^{(e)} + \bar{\boldsymbol{G}}_7^{(e)} + \bar{\boldsymbol{G}}_8^{(e)}$ *to the devices in group* 1 *which now have a share of the global aggregate* $\boldsymbol{G}^{(e)}$.

Notice that there is at most one solid incoming and outgoing edge at each node. This means that at any step devices receive at most one message and send at most one message to devices in another group. This way we avoid a congestion of the network and can collect the shares of the aggregates efficiently in group 1. Furthermore, although we picked N *to be a power of* 2, (II.24) *and* (II.25) *hold for any* N *that divides* n.

At any step, each device has access to at most one share of any $\bar{\boldsymbol{G}}_j^{(e)}$. Note that shares from different groups encode different gradients and can not be used together to extract any information. As a result, the privacy is not impaired by the grouping as we still need k colluding devices to decode any SSS, while the central server is only able to decode the global aggregate.

Both the grouping and the inter-group communication are detrimental to straggler mitigation compared to CodedSecAgg with only one group. However, as we show in the next section, the reduced decoding cost at the central server and the reduced communication cost in the first phase compensate for the reduced straggler mitigation.

7 Comparison of CodedPaddedFL and CodedSecAgg

The core idea behind CodedPaddedFL and CodedSecAgg is to introduce redundancy on the devices' datasets in FL without leaking additional information about the data to other devices. Subsequently, the redundancy can be leveraged at the central server to mitigate the straggler effect on the overall latency. However, the privacy goals of the two schemes are different. CodedPaddedFL retains the privacy level of conventional FL, meaning that i) no device gains information about other devices' datasets beyond what can be obtained from the global gradient, and ii) the central server only learns the local gradients at each device. On the other hand, CodedSecAgg has a much higher privacy goal. In particular, any set of at most z agents, including the central server, cannot infer any information about the datasets and local gradients pertaining to devices outside of the set.

CodedPaddedFL introduces redundancy via gradient codes and uses one-time padding to retain the privacy. To use gradient codes, each device needs access to α ($1 \leq \alpha \leq n$) datasets, which entails downloading $\alpha - 1$ other (one-time padded) datasets. In contrast, CodedSecAgg utilizes Shamir's SSS, i.e., a combination of Reed-Solomon codes and padding, to introduce redundancy and retain the privacy. Here, each device needs to download $n - 1$ other datasets. This means that the additional privacy of the devices' datasets is traded off with a potentially higher communication load in the sharing phase. Furthermore, the necessity to prevent

the central server from learning any local gradient puts additional limitations on the grouping in CodedSecAgg. In CodedPaddedFL, the central server can simply apply the scheme in each group separately and aggregate the gradients from all groups directly whereas in CodedSecAgg we need an evolved protocol to aggregate the gradients from different groups that involves multiple communication rounds.

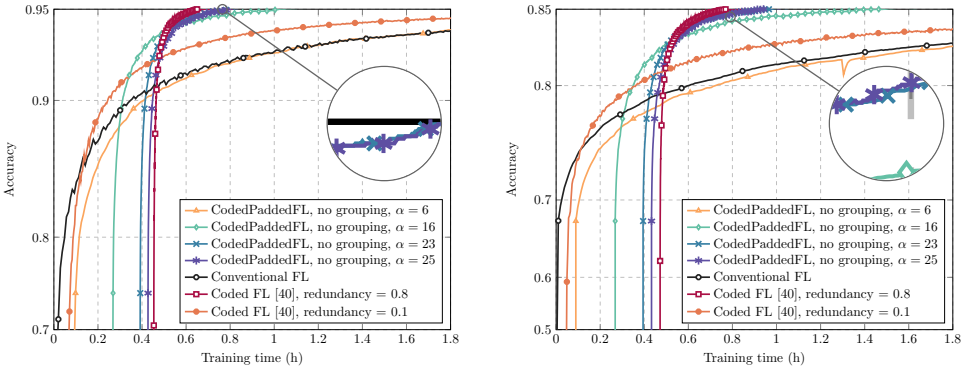
In conclusion, when model inversion attacks are of no concern, e.g., when the central server is trusted, CodedPaddedFL is the most efficient scheme. However, when the central server shall be prevented from learning local gradients, CodedSecAgg provides this additional privacy at a slightly higher communication cost.

8 Numerical Results

We simulate an FL network in which devices want to collaboratively train on the MNIST [45] and Fashion-MNIST [46] datasets, i.e., we consider the application of the proposed schemes to a classification problem. To do so, we preprocess the datasets using kernel embedding via Python's radial basis function sampler of the sklearn package (with 5 as kernel parameter and 2000 features). We divide the datasets into training and test sets. Furthermore, the training set is sorted according to the labels to simulate non-identically distributed data before it is divided into n equally-sized batches which are assigned without repetition to the n devices. We use $\ell = 48$ bits to represent fixed-point numbers with a resolution of $f = 24$ bits in CodedPaddedFL and CodedSecAgg, whereas we use 32-bit floating point numbers to represent the data for the schemes we compare with, i.e., conventional FL, the scheme in [40], and LightSecAgg. For our proposed schemes, we assume that the computation of the first local gradient and $\boldsymbol{X}^{(i)\mathsf{T}}\boldsymbol{X}^{(i)}$ happens offline because no interaction is required by the devices to compute those. We sample the setup times $_i$ at each epoch and assume that they have an expected value of 50% of the deterministic computation time. In particular, device i performing ρ_i MAC operations at each epoch yields $\eta_i = \frac{2\tau_i}{\rho_i}$. For the communication between the central server and the devices, we assume they use the LTE Cat 1 standard for IoT applications, which means that the corresponding rates are $\gamma^{\mathrm{d}} = 10$ Mbit/s and $\gamma^{\mathrm{u}} = 5$ Mbit/s. The probability of transmission failure is $p_i = 0.1$ and we add a 10% header overhead to all transmissions. For the learning, we use a regularization parameter $\lambda = 9 \times 10^{-6}$ and an initial learning rate of $\mu = 6.0$, which is updated as $\mu \leftarrow 0.8\mu$ at epochs 200 and 350.

8.1 Coded Federated Learning

We first consider a network with $n = 25$ devices. We model the heterogeneity by varying the MAC rates τ_i across devices. In particular, we have 10 devices with a MAC rate of $25 \cdot 10^6$ MAC/s, 5 devices with $5 \cdot 10^6$, 5 with $2.5 \cdot 10^6$, and the last 5 with $1.25 \cdot 10^6$, whereas the central server has a MAC rate of $8.24 \cdot 10^{12}$ MAC/s. These MAC rates are chosen in accordance with the performance that can be expected from devices with chips from the Texas Instruments TI MSP430 family[50]. For the conventional FL training, we perform mini-batch gradient descent where we use a fifth of the data at each epoch. We chose the mini-batch size as a compro-



(a) MNIST dataset

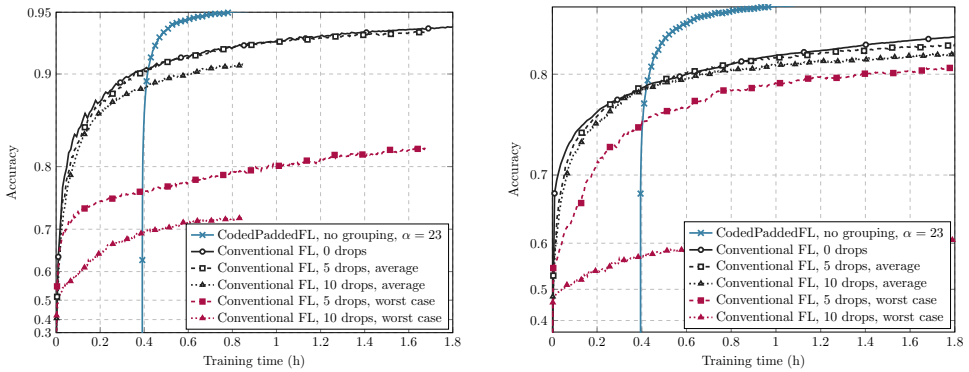
(b) Fashion-MNIST dataset

Figure II.10: Training time for the proposed CodedPaddedFL with different values of α, the coded FL scheme in [40], and conventional FL.

mise between the extreme cases: a low mini-batch size does not allow for much parallelization whereas a large mini-batch size might be exceeding the parallelization capabilities of the devices and thereby slow the training down. Note that for CodedPaddedFL, we train on $\boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}$ for which it does not give any benefits to train on mini-batches. Training on smaller batches can significantly speed up the gradient computation when the complexity depends on the dataset size as is the case for conventional FL. However, the pre-computation of $\boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}$ renders the gradient computation independent of the dataset size in CodedPaddedFL. The simulation implementation code is available at [51].

In Fig. II.10a, we plot the accuracy over the training time on the MNIST dataset for the proposed CodedPaddedFL with no grouping, i.e., for $N = 1$, for different values of $\alpha \in \{6, 16, 23, 25\}$, conventional FL, and the scheme in [40]. Note that $\alpha = 25$ corresponds to a replication scheme where all devices share their padded data with all other devices. By the initial offsets in the plot, we can see that the encoding and sharing, i.e., phase one, takes longer with increasing values of α. However, the higher straggler mitigation capabilities of high values of α result in steep curves. Our numerical results show that the optimal value of α depends on the target accuracy. For the considered scenario, $\alpha = 23$ reaches an accuracy of 95% the fastest. Conventional FL has no initial sharing phase, so the training can start right away. However, the lack of straggler mitigation capabilities result in a slow increase of accuracy over time. For an accuracy of 95%, CodedPaddedFL yields a speed-up factor of 6.6 compared to conventional FL. For levels of accuracy below 90%, conventional FL performs best and there are also some α, such as $\alpha = 6$, where the performance of CodedPaddedFL is never better than for conventional FL.

The scheme in [40] achieves speed-ups in training time by trading off the users' data privacy. In short, in this scheme devices offload computations to the central server through the parity data to reduce their own epoch times. The more a device is expected to straggle, the more it offloads to the central server. To quantify



(a) MNIST dataset

(b) Fashion-MNIST dataset

Figure II.11: Training time for the proposed CodedPaddedFL with $\alpha = 23$ and conventional FL with a subset of the fastest devices.

how much data is offloaded, the authors introduce a parameter called redundancy which lies between 0 and 1. A low value of redundancy means little data is offloaded and thereby leaked, whereas a high value of redundancy means that the central server does almost all of the computations and results in a high information leakage. We consider two levels of redundancy for our comparison, 0.1 and 0.8. We can see that for a redundancy of 0.1 our scheme outperforms the scheme in [40] for significant levels of accuracy whereas CodedPaddedFL is only slightly slower for a redundancy of 0.8. Note, however, that a redundancy of 0.8 means that the devices offload almost all of the data to the central server, which not only leaks the data but also transforms the FL problem into a centralized learning problem.

In Fig. II.10b, we plot the accuracy over training time for the Fashion-MNIST dataset with no grouping, i.e., for $N = 1$. We observe a similar qualitative performance. In this case, $\alpha = 25$ is the value for which CodedPaddedFL achieves fastest an accuracy of 85%, yielding a speed-up factor of 9.2 compared to conventional FL. For a target accuracy between 80% and 85%, CodedPaddedFL with different $\alpha < 25$ performs best.

8.2 Client Drift

In Figs. II.11a and II.11b, we compare the performance of CodedPaddedFL with $N = 1$ (i.e., no grouping) for $\alpha = 23$ with that of conventional FL where the 5 or 10 slowest devices are dropped at each epoch for the MNIST and Fashion-MNIST datasets, respectively. For conventional FL, we plot the average performance and the worst-case performance. Dropping devices in a heterogeneous network with strongly non-identically distributed data can have a big impact on the accuracy. This is highlighted in the figures; while dropping devices causes a limited loss in accuracy on average, in some cases the loss is significant. For the MNIST dataset, in the worst simulated case, the accuracy reduces to 82.4% for 5 dropped devices

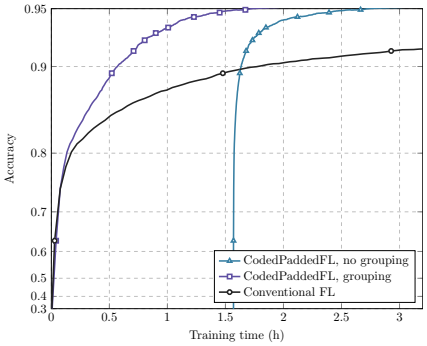


Figure II.12: Training on the MNIST dataset with CodedPaddedFL, with and without grouping the $n = 120$ devices.

and to 72.1% for 10 dropped devices (see Fig. II.11a), underscoring the client drift phenomenon. For the Fashion-MNIST dataset, the accuracy reduces to 82.7% and 61.5% for 5 and 10 dropped devices, respectively (see Fig. II.11b). The proposed CodedPaddedFL outperforms conventional FL in all cases; CodedPaddedFL has the benefit of dropping the slow devices in each epoch while not suffering from a loss in accuracy due to the redundancy of the data across the devices (see Proposition 3).

8.3 Grouping

The advantages of grouping the devices when training on the MNIST dataset are demonstrated in Fig. II.12. We now consider a network with $n = 120$ devices and draw their MAC rates uniformly at random from the set of available rates (25, 5, 2.5, and $1.25 \cdot 10^6$ MAC/s). For a given target accuracy, we minimize the training time over all values of α and number of groups N with the constraint that N divides n to limit the search space. For the baseline without grouping, we fix $N = 1$. As we can see, grouping significantly reduces the initial communication load of CodedPaddedFL. This is traded off with a shallower slope due to slightly longer average epoch times because of the reduced straggler mitigation capabilities when grouping devices. Nevertheless, the gains from the reduced time spent in phase one are too significant, and grouping the devices reduces the overall latency significantly for $n = 120$ devices. As a result, CodedPaddedFL with grouping achieves a speed-up factor of 18 compared to conventional FL for a target accuracy of 95% in a network with $n = 120$ devices.

8.4 Coded Secure Aggregation

In Fig. II.13 (left plot), we plot the training over time for our proposed CodedSecAgg and compare it with LightSecAgg for different number of colluding agents z when training on the MNIST dataset. We consider a network with $n = 120$ devices. As with CodedPaddedFL, our proposed scheme again distinguishes itself by the initial offset due to the sharing of data in phase one which quickly is made up for

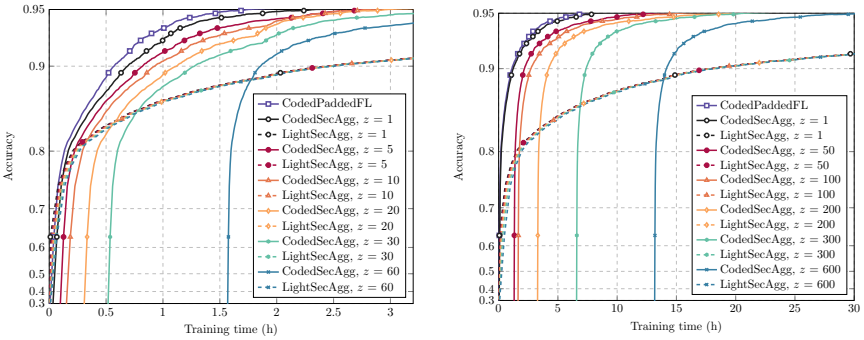


Figure II.13: Training on the MNIST dataset with CodedPaddedFL and CodedSecAgg with grouping in comparison to LightSecAgg for $n = 120$ (left plot) and $n = 1000$ (right plot) devices.

by a much reduced average epoch time due to the straggler mitigation in phase two. As a result, our proposed CodedSecAgg achieves a speed-up factor of 18.7 compared to LightSecAgg for a target accuracy of 95% when providing security against a single malicious agent.

We notice that CodedSecAgg is more sensitive to an increase in the security level z whereas LightSecAgg is almost unaffected regardless whether one desires to be secure against a single malicious agent or 60 colluding agents. A reason why our proposed scheme is more affected lies in the grouping: we require $k > z$, i.e., we require more than z devices in each group. As a result, a high value of z restricts the flexibility in grouping the devices. Nevertheless, even when we allow 60 agents to collude, i.e., half of the devices, our scheme achieves a speed-up factor of 6.6. Note that for $z = 60$, CodedSecAgg requires at least 61 devices per group. Given that there are only 120 devices in total and we require each group to have the same size, there is only one group of devices, i.e., no grouping, for $z = 60$.

We can also quantify the additional cost in terms of latency that secure aggregation imposes compared to CodedPaddedFL where the central server may learn the local models. For an accuracy of 95% on the MNIST dataset, to prevent a model inversion attack CodedSecAgg incurs a moderate additional 34% of latency compared to CodedPaddedFL.

In Fig. II.13 (right plot), we increase the number of devices in the network to $n = 1000$. Qualitatively little changes compared to the scenario in the left plot, which highlights the great scalability with the number of devices of our proposed scheme. However, we see a decrease in the additional relative latency due to secure aggregation. CodedSecAgg with a privacy level of $z = 1$ now needs only 15% more latency compared to CodedPaddedFL. The higher cost of the sharing phase of CodedSecAgg compared to CodedPaddedFL becomes negligible in the long run. Due to the large number of devices, the straggler effect becomes more severe and the epoch times become longer. In comparison, the slightly longer sharing phase of CodedSecAgg is barely noticeable. Compared to LightSecAgg, CodedSecAgg achieves a speed-up factor of 10.4 for 600 colluding agents, whereas the speed-up factor increases to 38.9 for a single malicious agent in the network.

We observe similar performance of CodedSecAgg on the Fashion-MNIST dataset, both compared to LightSecAgg and CodedPaddedFL.

9 Conclusion

We proposed two new federated learning schemes schemes, referred to as CodedPaddedFL and CodedSecAgg, that mitigate the effect of stragglers. The proposed schemes borrow concepts from coded distributed computing to introduce redundancy across the network, which is leveraged during the iterative learning phase to provide straggler resiliency—the central server can update the global model based on the responses of a subset of the devices.

CodedPaddedFL and CodedSecAgg yield significant speed-up factors compared to conventional federated learning and the state-of-the-art secure aggregation scheme LightSecAgg, respectively. Further, they converge to the global optimum and do not suffer from the client drift problem. While the proposed schemes are tailored to linear regression, they can be applied to nonlinear problems such as classification through kernel embedding.

An interesting topic for future work is to investigate how to adapt CodedPaddedFL and CodedSecAgg to nonlinear problems, e.g., through the use of Lagrange coding [52] and pice-wise linear functions. Furthermore, investigating ways to perform multiple local gradient updates in CodedPaddedFL and CodedSecAgg before aggregating the local gradients at the central server to reduce the communication load in the network is another interesting direction to pursue.

References

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artificial Intell. Stats. (AISTATS)*, Fort Lauderdale, FL, Apr. 2017, pp. 1273–1282.

[2] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop Private Multi-Party Mach. Learn. (PMPML)*, Barcelona, Spain, Dec. 2016.

[3] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020.

[4] A. Jochems *et al.*, "Developing and validating a survival prediction model for NSCLC patients through distributed learning across 3 countries," *Int. J. Radiat. Oncol. Biol. Phys.*, vol. 99, no. 2, pp. 344–352, Oct. 2017.

[5] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," in *Proc. Mach. Learn. Syst. (MLSys)*, Stanford, CA, Mar./Apr. 2019, pp. 374–388.

[6] Z. Charles and J. Konečný, "On the outsized importance of learning rates in local update methods," Jul. 2020, arXiv:2007.00878.

[7] A. Mitra, R. Jaafar, G. J. Pappas, and H. Hassani, "Linear convergence in federated learning: Tackling client heterogeneity and sparse gradients," in *Proc. Neural Inf. Process. Syst. (NeurIPS)*, online, Dec. 2021, pp. 14 606–14 619.

[8] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," Mar. 2019, arXiv:1903.03934.

[9] Y. Li, S. Yang, X. Ren, and C. Zhao, "Asynchronous federated learning with differential privacy for edge intelligence," Dec. 2019, arXiv:1912.07902.

[10] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization for heterogeneous networks," in *Proc. ICML Workshop Adaptive Multitask Learn. (AMTL)*, Long Beach, CA, Jun. 2019.

[11] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," in *Proc. Neural Inf. Process. Syst. (NeurIPS)*, Vancouver, Canada, Dec. 2020, pp. 7611–7623.

[12] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. Jarvis, "SAFA: A semi-asynchronous protocol for fast federated learning with low overhead," *IEEE Trans. Comput.*, vol. 70, no. 5, pp. 655–668, May 2021.

[13] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Denver, CO, Oct. 2015, pp. 1322–1333.

[14] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *Proc. IEEE Int. Conf. Comp. Commun. (INFOCOM)*, Paris, France, Apr./May 2019, pp. 2512–2520.

[15] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Dallas, TX, Oct./Nov. 2017, pp. 1175–1191.

[16] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramachandran, "FastSecAgg: Scalable secure aggregation for privacy-preserving federated learning," in *Int. Workshop Fed. Learn. User Privacy Data Confidentiality*, Vienna, Austria, Jul. 2020.

[17] J. So, B. Güler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 1, pp. 479–489, Mar. 2021.

[18] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly)logarithmic overhead," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, online, Nov. 2020, pp. 1253–1269.

[19] A. R. Elkordy and A. S. Avestimehr, "HeteroSAg: Secure aggregation with heterogeneous quantization in federated learning," *IEEE Trans. Commun.*, vol. 70, no. 4, pp. 2372–2386, Apr. 2022.

[20] J. So, C. He, C.-S. Yang, S. Li, Q. Yu, R. E. Ali, B. Güler, and S. Avestimehr, "LightSecAgg: a lightweight and versatile design for secure aggregation in federated learning," in *Proc. Mach. Learn. Syst. (MLSys)*, Santa Clara, CA, Aug./Sep. 2022.

[21] Y. Zhao and H. Sun, "Information theoretic secure aggregation with user dropouts," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Melbourne, Australia, Jul. 2021, pp. 1124–1129.

[22] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and verifiable federated learning," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 911–926, 2020.

[23] T. Jahani-Nezhad, M. A. Maddah-Ali, S. Li, and G. Caire, "SwiftAgg: Communication-efficient and dropout-resistant secure aggregation for federated learning with worst-case security guarantees," Feb. 2022, arXiv:2202.04169.

[24] A. R. Chowdhury, C. Guo, S. Jha, and L. van der Maaten, "EIFFeL: Ensuring integrity for federated learning," Dec. 2021, arXiv:2112.12727.

[25] T. Jahani-Nezhad, M. A. Maddah-Ali, S. Li, and G. Caire, "SwiftAgg+: Achieving asymptotically optimal communication load in secure aggregation for federated learning," Mar. 2022, arXiv:2203.13060.

[26] J. So, R. E. Ali, B. Güler, and A. S. Avestimehr, "Secure aggregation for buffered asynchronous federated learning," in *1st NeurIPS Workshop New Frontiers Fed. Learn. (NFFL)*, online, Dec. 2021.

[27] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," in *Proc. Int. Conf. Artificial Intell. Stats. (AISTATS)*, online, Mar. 2022, pp. 3581–3607.

[28] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Washington, DC, Dec. 2016.

[29] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Proc. Neural Inf. Process. Syst. (NIPS)*, Long Beach, CA, Dec. 2017, pp. 4403–4413.

[30] K. Lee, M. Lam, R. Pedersani, D. Papailiopoulos, and K. Ramachandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

[31] A. Severinson, A. Graell i Amat, and E. Rosnes, "Block-diagonal and LT codes for distributed computing with straggling servers," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 1739–1753, Mar. 2019.

[32] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Trans. Inf. Theory*, vol. 65, no. 7, pp. 4227–4242, Jul. 2019.

[33] S. Dutta, V. Cadambe, and P. Grover, ""Short-Dot": Computing large linear transforms distributedly using coded short dot products," *IEEE Trans. Inf. Theory*, vol. 65, no. 10, pp. 6171–6193, Oct. 2019.

[34] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Trans. Inf. Theory*, vol. 66, no. 1, pp. 278–301, Jan. 2020.

[35] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Sydney, Australia, Aug. 2017, pp. 3368–3376.

[36] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Proc. Neural Inf. Process. Syst. (NIPS)*, Long Beach, CA, Dec. 2017, pp. 5440–5448.

[37] R. Schlegel, S. Kumar, E. Rosnes, and A. Graell i Amat, "Privacy-preserving coded mobile edge computing for low-latency distributed inference," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 3, pp. 788–799, Mar. 2022.

[38] J. Zhang and O. Simeone, "On model coding for distributed inference and transmission in mobile edge computing systems," *IEEE Commun. Lett.*, vol. 23, no. 6, pp. 1065–1068, Jun. 2019.

[39] A. Frigård, S. Kumar, E. Rosnes, and A. Graell i Amat, "Low-latency distributed inference at the network edge using rateless codes," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, Berlin, Germany, Sep. 2021.

[40] S. Prakash, S. Dhakal, M. R. Akdeniz, Y. Yona, S. Talwar, S. Avestimehr, and N. Himayat, "Coded computing for low-latency federated learning over wireless edge networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 233–250, Jan. 2021.

[41] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should I trust you?" Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining (KDD)*, San Francisco, CA, Aug. 2016, pp. 1135–1144.

[42] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Sydney, Australia, Aug. 2017, pp. 3145–3153.

[43] E. Tjoa and C. Guan, "A survey on explainable artificial intelligence (XAI): Toward medical XAI," *IEEE Trans. Neural Network Learn. Syst.*, vol. 32, no. 11, pp. 4793–4813, Nov. 2021.

[44] IBM. (2022). [Online]. Available: https://www.ibm.com/se-en/topics/linear-regression

[45] Y. LeCun, C. Cortes, and C. J. C. Burges, "The MNIST database of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist

[46] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," Aug. 2017, arXiv:1708.07747. [Online]. Available: https://research.zalando.com/project/fashion_mnist/fashion_mnist

[47] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[48] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *Proc. Int. Conf. Financial Crypto. Data Secur. (FC)*, Tenerife, Spain, Jan. 2010, pp. 35–50.

[49] C. E. Shannon, "Communication theory of secrecy systems," *The Bell Syst. Tech. J.*, vol. 28, no. 4, pp. 656–715, Oct. 1949.

[50] Texas Instruments. MSP430 microcontrollers. [Online]. Available: https://www.ti.com/microcontrollers-mcus-processors/microcontrollers/msp430-micrcontrollers/overview.html

[51] R. Schlegel. (2022). [Online]. Available: https://github.com/ReentSchlegel/CodedPaddedFL-and-CodedSecAgg-Straggler-Mitigation-and-Secure-Aggregation-in-Federated-Learning

[52] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. Int. Conf. Artificial Intell. Stats. (AISTATS)*, Okinawa, Japan, Apr. 2019, pp. 1215–1225.

Graphic design: Communication Division, UiB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230860120 (print)
9788230859469 (PDF)